REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-01-

0288

1. REPORT DATE *(DD-MM-YYYY)* 31-01-2001	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* 1 Nov 97 - 31 Oct 00
4. TITLE AND SUBTITLE Fusion as an Operation on Formal Systems: A Formal Framework for Information Fusion		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER F49620-98-1-0043
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Mieczyslaw M. Kokar, Jerzy Weyman, Jerzy A. Tomasik		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Northeastern University 360 Huntington Avenue Boston, MA 02115		**8. PERFORMING ORGANIZATION REPORT NUMBER** NEU-COE-ECE-MMK-2001-01
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM 801 North Randolph Street Arlington, VA 22203-1977		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFOSR)
NOTICE OF TRANSMITTAL DTIC. THIS TECHNICAL REPORT
HAS BEEN REVIEWED AND IS APPROVED FOR PUBLIC RELEASE
LAW AFR 190-12. DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This project was focused on a formal framework for analyzing and developing information fusion systems. Category theory was selected as the formal basis for specifying information fusion systems. A formal definition of "information fusion" was introduced. Various kinds of fusion were defined, their properties and inter-relationships analyzed. An initial ontology of information fusion was presented. A procedure for developing information fusion systems using formal methods was proposed. Examples of reasoning about measures of effectiveness of information fusion systems were shown. Towards this goal fuzzy logic was formally specified in the Slang language and properties of information fusion were proved. This showed that within this formal framework one can reason about the system being developed in the specification and design phases, rather than after deployment. Finally, the selection of wavelet based features for ATR was investigated within the formal framework.

15. SUBJECT TERMS

Information Fusion, Formal Methods, Design Framework, Logic, Category Theory

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		181	**19b. TELEPHONE NUMBER** *(include area code)*

FINAL REPORT

Project title:

Fusion as an Operation on Formal Systems: A Formal Framework for Information Fusion

Grant no: F49620-98-1-0043

AFOSR/NM

PI: Prof. Mieczyslaw M. Kokar
Department of Electrical and Computer Engineering

Northeastern University
360 Huntington Avenue
Boston, MA 02115

April 9, 2001

Investigators:

Professor Mieczyslaw M. Kokar (PI)

Professor Jerzy Weyman (Co-PI)

Professor Jerzy A. Tomasik (Co-PI)

Hongge Gao (Ph.D. student)

Jiao Wang (Ph.D. student)

Gulsah Cakiroglou (M.S. student)

Jingsong Li (M.S. student)

Marek Malczewski (M.S. student)

Contents

Chapter 1

Project Overview

In spite of a significant progress in research on information fusion, there is still a lack of a formal theoretical framework for integrating disparate sources of information, and especially for developing such systems. The consequence of this situation is that the designer of an information fusion system does not have the necessary theoretical or computer-based tools to analyze various design solutions with respect to a given system performance measure. In particular, the designer cannot reason about the impact of uncertainty of information (e.g., signal to noise ratio (SNR) or imprecision of a definition of a concept) on the uncertainty of the decisions derived by the system. For instance, the designer is interested in the answer to the question of what is the accuracy of a given target recognition decision given particular levels of uncertainty of information received by a number of sensors. Also, the designer could use a computer aided software engineering (CASE) tool to support him or her in the process of software development. While a number of CASE tools are available from various vendors, they do not provide enough rigor to ensure that a system specification is correctly translated into the design representation and then into a correct code implementation. The main objective of this project is to develop a formal information fusion framework for analyzing and developing information fusion systems.

The characteristics of a formal approach include:

1. The use of logic and mathematics to formulate knowledge of: requirement specifications of fusion systems (including performance measures); theories of the sensed world; theories of sensors.

2. The use of this formal knowledge in the development of fusion systems.

3. The use of formal (provably correct) transformations in the process of development of fusion systems,.

4. The use of formal method tools (theorem provers) in the development process.

To address the problems mentioned above within the formal method paradigm we set the goal of this project as the development of a Formal Information Fusion Framework (FIFF) within which such problems could be precisely defined and then addressed. More specifically, our goal is to develop the following components for such a framework.

1. A basic language for representing theories necessary for analyzing and developing information fusion systems, including the basic vocabulary for fusion system elements, system architecture, main functions, typical goals, and selected problem and world representations.

2. A host of basic fusion theories related to fusion of information, e.g., theories for sensing, theories for particular sensors, theories for data association, theories for refinement of system requirements and propagation of performance measures, theories for combining data into one coherent structure, and some theories of the physical world.

3. Basic procedures for specifying, designing, implementing information fusion systems. In particular, it should contain procedures for checking consistency of proposed solutions with the selected theories of the world, procedures for checking completeness and internal consistency of particular representations, procedures for refining representations that preserve specified performance criteria of the fusion system.

Our approach consists of three main components:

1. Formulation of the information fusion problem.

2. Selection of category theory as a theory for defining and analyzing fusion operators.

3. Selection of the Specware formal specification tool as our CASE tool.

To formalize the concept of information fusion we chose the discipline of *model theory*, a sub-discipline of formal mathematical logic. We used model theory to formalize knowledge about sensors, objects, the world, and any necessary background knowledge. Each of these components of knowledge is represented by a collection of theories and classes of their models. Theories are syntactical objects, while models provide the semantics for these objects. A fusion operator takes a number of theories and their models and produces a fused theory and a class of its models. Therefore, in our approach, unlike in any other approach, fusion is an operation on structures, rather than just on data. Operations on data are obviously performed by an information processing system, but the essence of fusion is captured in the system specification and design stages.

We stipulated that the fusion operator must be *sound* and *correct*. The soundness is expressed as a requirement that the fused class of models be models of both the fused theory and of the background knowledge. The correctness is expressed as a requirement that the fused theory derive correct decisions (possibly generalized).

We use *category theory* as a unifying concept for formally developed information fusion systems. Category theory is a mathematically sound representation technique used to capture the commonalities and relationships between objects; in our case these objects are structures - *theories* and *models*. This feature makes category theory a very elegant language for describing information fusion systems and the information fusion process itself. It seems the only theory that can capture the combination (fusion) of structures like models and theories. Additionally, it provides the operator of *colimit* that allows us to combine such structures.

To investigate the formal approach to information fusion we selected the SpecwareF^{TM} specification tool (from Kestrel Institute). Specware is based on category theory. In Specware, category theory objects are called specs (specifications). Specware supports the colimit operation. It also supports progressive modular development of specifications. Additionally, it supports the process of refinement - the process of progressive translation of specifications into code. The refinement process is guaranteed to be correct.

1.1 Summary of Results

During the time of this project we have accomplished the goals described above. We have already submitted a number of papers describing our results and still are in the process of writing more papers that document our findings. In the papers we have also included problems that we have identified in this research and which, we believe, should be undertaken as future research topics.

Formalization of the Information Fusion Problem

Formalization of the information fusion problem was the main goal of this project. We present our solution in Chapter 2. We believe that this formalization will constitute a foundation for the theory of information fusion. We have identified some basic vocabulary that constitutes the vocabulary of the theory of information fusion. We formalized the vocabulary as theories in Specware, with information objects being sorts with associated axioms, and relations/functions as operations with associated axioms.

- Within our formal framework we were able to define such basic notions like *data fusion* and *decision fusion.* These notions are widely used in the fusion literature, however, nowhere else are they precisely defined.
- In our framework we were able to define the notion of *category of information fusion systems.* Within this category we formulated the notion of *subclass*, i.e., that a given fusion system is a subclass of another kind of fusion system.
- The notion of subclass allowed us to consider relationships among classes of fusion systems. In particular, we were able to prove that the class of decision fusion systems is a subclass of data fusion systems.
- We investigated the question of when decision fusion is equivalent to data fusion. In particular, we formulated criteria that need to be satisfied for decision fusion to be equivalent to data fusion. We gave sufficient and necessary conditions for the equivalence.
- Within our formal framework we were able to define some special classes of fusion systems, e.g., *monotonic, consistent, preferential.* These classes are not explicitly used in the fusion literature, but since they constitute special categories, their specific properties may be used in the analysis of fusion systems.

An Ontology for Information Fusion

We have developed various specifications (in Slang, the language of Specware) that can be used in the process of developing new fusion systems. These specifications constitute an initial *ontology* for information fusion. While some of the specifications were developed with the goal of investigating a particular research questions, and thus are more of an experimental nature, some others capture main characteristics of a practical nature. For instance, we have developed a number of theories for sensors. The list of theories covers both real sensors, especially various kind of radars, e.g., the Synthetic Aperture Radar, and some artificial sensors that we used for demonstration of our ideas, like artificial 1D range and intensity sensors. This ontology could be extended to become of practical size and value. The ontology is composed of two major components: theories of geometry and theories of sensors. Additionally, the ontology includes templates of goals.

A Procedure for the Development of Information Fusion Systems Based on Formal Methods

As specified in the goals of this project, we developed a procedure for developing information fusion systems using formal methods. This procedure was partially implemented and tested. The final result will is a demo that exemplifies advantages of using the formal approach to designing information fusion systems.

To accomplish this goal we first developed a use case (in UML: the Unified Modeling Language) that describes the interaction of the actor (information fusion designer) with the system (Formal Information Fusion Framework). The use case steps include Goal Specification, Sensor Specification, and Specification Synthesis. Specification Synthesis, in turn, includes Ontology Selection and Search, as well as Theorem Proving. Theorem Proving serves two purposes: consistency checking of the specification being developed and proving that the specification, and consequently the system that can be developed from the specification using formal methods, would have particular properties. Our framework includes templates that are used to support these steps.

In addition to the use case, we tested these procedures on two examples in which we developed specifications of target recognition systems, one for a simple scenario of recognition of rectangles and triangles, and another one of a more realistic nature that uses radars. Using these specifications we analyzed various design solutions and proved their properties. The results of these experiments will appear as a Ph.D. thesis which is expected to be complete by the end of this year. A demo is available at our project's web page.

Reasoning in the Design Phase

Since the most important application-oriented goal of this effort was to show that the designer of an information fusion system can reason about the fusion system s/he is designing before it actually is implemented, we devoted much of attention to the issue of Measures of Effectiveness (MOE) and the propagation of the expected MOE through the phases of the system development process. Our focus was on the MOEs that use fuzzy measures and fuzzy reasoning.

We conducted the studies of fuzzy measures in two directions. One was the development of a library of formal specifications for fuzzy logic. The second was the theoretical study of the semantics for fuzzy variables.

We proposed to use fuzzy set theory (fuzzy numbers, and fuzzy operators) to characterize the uncertainty of input information and then manipulate (reason about) the uncertainty of decisions based upon the input information. A library of specifications of fuzzy set theory was developed using category theory and Specware. The library was then used to construct specifications of fuzzy information processing systems. As an example, a fuzzy edge detection algorithm was specified in Specware. The algorithm uses fuzzy operations in its processing. Then we formulated some theorems in the language of Specware (Slang). As a next step we proved the theorems. Ideally, the proofs should be automatic. However, due to some shortcomings of the current theorem provers, we had to use a hybrid approach, i.e., the prover had to be guided through the proof. The main use of the prover was to confirm the conjectures generated by the human.

To achieve the second goal of this sub-effort, we conducted extensive literature studies related to the semantics of fuzzy logic. This direction was initiated after we received input from the Air Force indicating that the Air Force in particular, and the DoD in general, may be reluctant to apply fuzzy logic due to the lack of the semantics of fuzzy logic. The result of our literature search partially confirmed this claim. We found many articles describing formalizations of various fuzzy logics. We concluded that there exist good formalizations of fuzzy logic, i.e., formalizations of the fuzzy inference rules. The second conclusion was that there is no formalization of the notion of fuzzy variable. In all research and application articles authors assume that the meaning of fuzzy variables is given, in most cases by an expert. In other words, authors accept the fact that the meaning of fuzzy variables is subjective. As a result of this study, we proposed an approach to the formalization of the semantics of fuzzy variables. The main idea behind this approach is to use unsupervised learning. This topic was identified as a topic for future research; a proposal has been submitted to the AFOSR.

Wavelet-Based Features

We continued our research on wavelet-based features. In this research, we showed a step-by-step approach to designing an sensor fusion system for ATR using wavelet coefficients as features. First, we showed how our former results can be incorporated into our formal framework. Towards this aim we showed the mapping of our system to the category theory objects, morphisms and refinements. One part of information required for this approach is a collection of theories of targets. In our earlier research, both the theories and the selection of features were done manually by the system designer. In this research effort we investigated the possibility of generating both the target theories and the features automatically, using a test database of target signatures. In particular, we proposed an approach to ATR in which we use relations among wavelet coefficients in the classification decision. The set of wavelet coefficients used for this purpose is not fixed, it is selected during the classification. The classification decision is made based upon the satisfaction of a relation by the wavelet coefficients. The relation is learned by the system automatically based upon a pre-classified database of target signatures. We implemented a simulation of such a system and compared its performance to a system in which a fixed set of wavelet coefficients was selected using an entropy based measure. The results of this comparison showed that the dynamically selected features and the learned relations gave a much better classification accuracy than the fixed set of features.

Decision Fusion

Although most of our research was focused on data fusion, we also investigated the issue of decision fusion. In particular, we investigated the case of decision fusion when two (or more) sensors and the fusion center have a common language to represent queries and decisions, while each of the sensors has its own interpretation of the formulas of the language. Fusion is achieved through the model-theoretic operation of direct product of models. Since not all (most) formulas are not preserved under the product we need a decision procedure to tell us how to combine decisions from particular sensors into one fused decision. Towards this aim the notion of Galvin system was used. The validity of the solution is formally defined and proved in an appropriate theorem. The main advantages of the proposed approach are that the decision mechanism is generic, i.e., it can check the validity of any goal formula, and that it is provably correct.

Chapter 2

Fusion as an Operation on Formal Systems

In this chapter we present a formalization of the notion of "information fusion" within the framework of formal logic and category theory. Within this framework information fusion systems can be specified in precise mathematical terms allowing in this way to formally reason about such specifications, designs and implementations. The notion of fusion proposed here differs from other approaches, where either data or decisions are fused. Here, the structures that represent the meaning of information (theories and models) are fused, while data are then simply processed using these structures (filtered through these structures). Within this framework the requirement of consistency of representations is formally and explicitly specified and then can be manipulated by the computer using automatic reasoning techniques.

We believe that this formalization will constitute a foundation for the theory of information fusion. We have identified some basic vocabulary that constitutes the vocabulary of the theory of information fusion. We formalized the vocabulary as theories in Specware, with information objects being sorts with associated axioms, and relations/functions as operations with associated axioms. We provide precise formal definitions of such notions like *data fusion* and *decision fusion*. These notions are widely used in the fusion literature, however, nowhere else are they precisely defined. We also define the notion of *category of information fusion systems*. Within this category we formulated the notion of *subclass*, i.e., that a given fusion system is a subclass of another kind of fusion system. The notion of subclass was then used to analyze relationships among classes of fusion systems. In particular, we prove that the class of decision fusion systems is a subclass of data fusion systems. We also formulated criteria that need to be satisfied for decision fusion to be *equivalent* to data fusion. We gave sufficient and necessary conditions for the equivalence. We also define some special classes of fusion systems, e.g., *monotonic, consistent, preferential.* These classes are not explicitly used in the fusion literature, but since they constitute special categories, their specific properties may be used in the analysis of fusion systems.

2.1 An Information Fusion System

An *information fusion system (IFS)* (see Figure 2.1) may receive inputs from various sources: sensors, data bases, knowledge bases, and other systems (over communication lines). In our discussion we will focus on inputs from sensors, since other sources of information can be considered as special kinds of sensors. Sensors may include vision cameras, infrared cameras, radars, accelerometers, ultrasound. Sensors provide measurements of a number of interrelated variables (*n-tuples*). In mathematical sense, sensors output either functions or relations. Sensor measurements may be interpreted as containing some degree of randomness. The sensors are controlled by the SDFS according to a SDFS *goal.* The *goal* is one of the external inputs to the SDFS provided by the human, while controls are one kind of IFS outputs. Typically, an IFS provides output screens that serve to communicate with the human operator (human-in-the-loop).

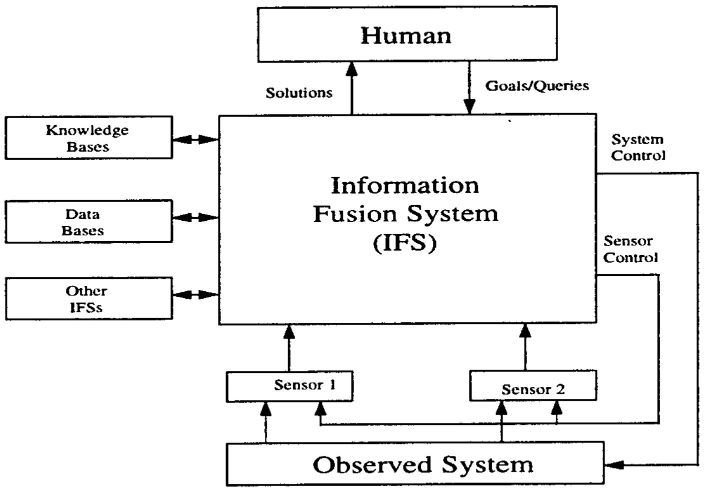

Figure 2.1: Information Fusion System (IFS)

In general, the goal of a SDFS is to interpret data received through sensors. It is expressed in a prespecified *goal language* understandable to either the user or another system. For instance, the goal can be to reconstruct the 3-D scene. In such a case the goal language includes a set of quadruples, where each element of the set represents the XYZ-coordinates of the world and the value of the light intensity at that point. In a tracking application, on the other hand, the goal is to find the state vector of the tracked object, and therefore the goal language includes six-tuples representing X, Y, and Z positions, as well as their first derivatives (velocities). If the goal is *object recognition*, then the goal language describes types of objects. In a *situation assessment* scenario, the language can be much more complex, and include descriptions of possible courses of events, consequences of particular actions, configurations of objects, features associated with objects and configurations, behaviors, etc. In the sensor fusion literature the goal language is often called *world model.*

A natural requirement for an information fusion system is that the interpretation of the data be "correct". Intuitively, this means that the objects identified by the IFS really exist in the world, that these objects have the features as identified by the IFS, that the relations recognized by the IFS really exist in the world, and that the interpretation does not violate the constraints that the world is known to obey, e.g., the laws of physics. In order to maintain the truthfulness of the interpretation, the system (*agent*) must maintain *consistency* of its representation.

To deal with the issue of correctness of interpretations we use the framework of model theory [6]. In particular, we make use of *formal languages* to describe the world and the sensing process and *models* to represent sensor data, operations on data, and relations among the data. Models consist of *carriers* of different sorts (usually sets) and many-sorted *operations*, and *relations* among the elements of different carriers. We use *theories* to represent symbolic knowledge about the world and about the sensors.

Fusion is then treated as a goal-driven operation of combining a fixed number of languages, theories and classes of models related to the goal, the sensors and the background knowledge, into one combined language, one combined theory and one combined class of models of the world. For languages, the combination involves extending the language so that it includes the well-formed forms (terms and formulas) related to the goal, sensors and background theories. For the fusion of theories, this usually involves combining the theorems of the theories, although one can also think of combining rules of inference. Fusion of classes of models is strictly related to fusion of theories (in general, it is a dual operation). Particular models can be fused by combining carriers, operations, and relations of the models into new carriers, operations and relations of the *fused model.* Therefore, fusion is a *formal system operator* that has multiple languages, theories and classes of models for inputs and a single language, a theory, and a class of models as the output.

The justification for fusion comes from the fact that the fusion system performs better, in relation to a prespecified performance measure, than any subsystem involving only one sensor. This may be either an added capability of answering queries about the world that cannot be answered using only one sensor, or increased precision of the answer, or an improvement of any other prespecified measure.

This understanding of fusion differs from more traditional approaches [13, 27], where issues like consistency are not dealt with explicitly. Rather, there is an underlying presumption that the operations of fusion are implemented in a consistent way by the human. In our approach, on the other hand, a framework is provided in which the requirement of consistency of representations can be formally and explicitly specified and then can be manipulated by the computer using automatic reasoning techniques.

Our approach is strongly motivated by software engineering, because an information fusion system is essentially a program. From the software engineering point of view, there exists a need for formal frameworks, in which systems can be represented, checked, and verified. Formal approaches provide more credibility for the system structure, and more predictability of its dynamical behavior. The use of formal methods brings more rigor to the development of complex computer systems[41]. Our goal is to develop formal specification methods oriented specifically towards sensor data fusion. Although specialized architectures have been proposed, for instance [38, 29], approaches to formal specifications of IFS systems, to our best knowledge, are not known in the literature.

From the programming languages point of view we treat our IFS system as a network of objects (in the object oriented paradigm) communicating via messages. Each object is an instance of a class. The goal for fusion is then to find classes from which, through class construction and instantiation, a whole SDF system can be built.

The main goal of our research in sensor data fusion (SDF) is to develop a systematic approach (a formal methodology) to specifying, designing, and implementing software systems for interpreting sensory information and for reasoning about the situation based upon this information and upon available data bases and knowledge bases. Towards this aim, we pursue several subgoals: (1) Development of a theory of sensor/data fusion. This includes formalization of the main concepts of the sensor data fusion theory in terms of *theories* and *classes of models.* (2) Development of a framework for specifying, designing, implementing and testing sensor/data fusion software systems. The basic tools utilized in this approach are computer-aided software engineering (CASE) tools and automatic theorem provers. (3) Development of algorithms for sensor data fusion. (4) Testing sensor data fusion approaches on real tasks and real data.

Here we deal primarily with the first of the issues mentioned above – the theory of sensor data fusion. Although there are several definitions of "fusion" in the subject literature, there does not seem to be an agreement on what is and what is not fusion. Our main objective is to contribute to the understanding of this notion. This understanding is necessary for specifying, designing and implementing information fusion systems.

2.2 First-level Decomposition of the IFS

In this presentation we follow a top-down approach by progressively decomposing the problem of development of an IFS into simpler sub-problems. In the first cut we decompose the IFS into three subsystems – Specification Synthesis, Code Generation (Refinement) and Information Processing – as shown in Figure 2.2. This decomposition follows the formal approach to software development, where code is developed in the process of progressive refinement of a formal software specification. The third block in Figure 2.2 represents the actual running system that takes inputs from all the sources and produces outputs in real time. Although it would be desirable to have the first two blocks also working in real time, it does not seem to be feasible at this time. Rather both Specification Synthesis and Code Generation are performed off-line. These two blocks, nevertheless, are shown as part of an IFS, since the main fusion problem, as presented here, is solved in the first block, i.e., in Specification Synthesis. This is essentially the only block where expertise of sensors and scenaria is needed. Code Generation can be performed independently of such expertise.

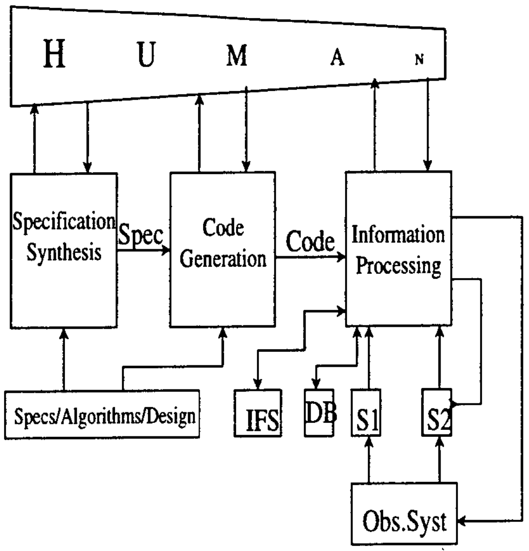

Figure 2.2: Information Fusion System: First-level Decomposition

2.3 Formalization of Information Fusion

2.3.1 Introduction

In spite of a significant progress in research on information fusion, there is still a lack of a formal theoretical framework for integrating disparate sources of information, and especially for developing such systems. The consequence of this situation is that the designer of an information fusion system does not have the necessary tools to analyze various design solutions with respect to a given system performance measure. In particular, the designer cannot reason about the impact of uncertainty of information (e.g., signal to noise ratio (SNR) or imprecision of a definition of a concept) on the uncertainty of the decisions derived by the system. For instance, the designer is interested in the answer to the question of what is the accuracy of a given target recognition decision given particular levels of uncertainty of information received by a number of sensors. Also, the designer could use a computer aided software engineering (CASE) tool to support him or her in the process of software development. While a number of CASE tools are available from various vendors, they do not provide enough rigor to ensure that a system specification is correctly translated into the design representation and then into a correct code implementation.

The main objective of our research is to develop a formal information fusion framework for analyzing and developing information fusion systems. The characteristics of a formal approach include:

1. The use of logic and mathematics to formulate knowledge of:

   (a) requirement specifications of fusion systems (including performance measures)
   (b) theories of the sensed world
   (c) theories of sensors

2. The use of this formal knowledge in the development of fusion systems

3. The use of formal (provably correct) transformations in the process of development of fusion systems

4. The use of formal method tools (theorem provers) in the development process.

More specifically, our goal is to develop the following components for such a framework.

1. A basic language for representing theories necessary for analyzing and developing information fusion systems, including the basic vocabulary for fusion system elements, system architecture, main functions, typical goals, and selected problem and world representations.

2. A host of basic fusion theories related to fusion of information, e.g., theories for sensing, theories for particular sensors, theories for data association, theories for refinement of system requirements and propagation of performance measures, theories for combining data into one coherent structure, and some theories of the physical world.

3. Basic procedures for specifying, designing, implementing information fusion systems. In particular, it should contain procedures for checking consistency of proposed solutions with the selected theories of the world, procedures for checking completeness and internal consistency of particular representations, procedures for refining representations that preserve specified performance criteria of the fusion system.

Our approach consists of three main components:

1. Formulation of the information fusion problem

2. Selection of category theory as a theory for defining and analyzing fusion operators

3. Selection of the Specware formal specification tool as our CASE tool.

To formalize the concept of information fusion we chose the discipline of *model theory*, a sub-discipline of formal mathematical logic. We use model theory to formalize knowledge about sensors, objects, the world, and any necessary background knowledge. Each of these components of knowledge is represented by a collection of *theories* and classes of their *models*. Theories are syntactical objects, while models provide the semantics for these objects. A fusion operator takes a number of theories and their models and produces a fused theory and a class of its models. Therefore, in our approach, unlike in any other approach, fusion is an operation on structures, rather than just on data. Operations on data are obviously performed by an information processing system, but the essence of fusion is captured in the system specification and design stages. We stipulated that the fusion operator must be *sound* and *correct*. The soundness is expressed as a requirement that the fused class of models be models of both the fused theory and of the background knowledge. The correctness is expressed as a requirement that the fused theory derive correct decisions (possibly generalized).

We are using category theory as a unifying concept for formally developed information fusion systems. Category theory is a mathematically sound representation technique used to capture the commonalities and relationships between objects; in our case these objects are structures - theories and models. This feature makes category theory a very elegant language for describing information fusion systems and the information fusion process itself. It seems the only theory that can capture the combination (fusion) of structures like models and theories. Additionally, it provides the operator of *colimit* that allows us to combine such structures.

To investigate the formal approach to information fusion we selected the Specware specification tool (Kestrel Institute). Specware is based on category theory. In Specware, category theory objects are called specs (specifications). Specware supports the colimit operation. It also supports progressive modular development of specifications. Additionally, it supports the process of *refinement* - the process of progressive translation of specifications into code. The refinement process is guaranteed to be correct.

We have conducted both theoretical and practical investigations. As a first step we focused on the formulation of the fusion problem in terms of category theory. This formulation is the subject of this paper.

2.4 The Fusion Problem

We formalize the information fusion problem in terms of triples (Σ, T, M), where Σ are *signatures* (languages), T - *theories* over the signatures, and M - classes of models of the theories. Signatures have the following form: $\Sigma = (\sigma, F)$, where σ are *sorts* and F are functions over the sorts. Theories are associated with the signatures; they are collections of axioms over the signatures. Signatures and associated theories are called *specifications*, or for short *specs*, $S = (\Sigma, T)$. Specs are considered as *objects* in the category *Sign* related through *morphisms*. Specs and morphisms are represented as *diagrams*. Models are .... We always assume that our theories are *consistent*, i.e., that they have models, formally denoted as $M \models T$.

We distinguish two kinds of information fusion – *data fusion* and *decision fusion*.

2.4.1 Data Fusion

The goal of data fusion is to develop a spec S_f and a fused class of models $\{M_f\}$, as described below. The inputs to this fusion process are some or all of the following specifications:

$$S_w = ((X, E, \Delta : X \to E), T_w)$$

$$S_1 = ((X_1, V_1, f_1 : X_1 \to V_1), T_1)$$

$$S_2 = ((X_2, V_2, f_2 : X_2 \to V_2), T_2)$$

$S_c = (C = \{C_1, C_2, ...\})$ - a spec of distinguished sorts (*coordinate sorts*)

$$\begin{tikzcd}
 & S_f & \\
S_1 \arrow[ur] & S_w \arrow[u] & S_2 \arrow[ul] \\
 & S_c \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.3: Data Fusion

The goal of information fusion is to find the following diagram $\mathcal{D}$ - a diagram of relations among the specs, where S_f is a specification:

$$S_f = ((X, E, \Delta : X \rightarrow E, X_1, V_1, X_2, V_2, f_1 : X_1 \rightarrow V_1, f_2 : X_2 \rightarrow V_2, D_f : (X_1 \rightarrow V_1), (X_2 \rightarrow V_2) \rightarrow (X \rightarrow 2^E)), T_f)$$

satisfying the conditions:

M_f, s.t. $M_f \models T_f$

$T_f \vdash \forall_{x \in X}\ \Delta(x) \in D_f(f_1, f_2)(x)$

2.4.2 Discussion

In the above formulation of the fusion problem we assume that S_w specifies the world that both sensors observe. In this specification, X represents the world coordinates, E is the objects in the world; they "occupy" locations in the world. The function Δ assigns these objects to particular locations. We do not assume that we know this function, but we assume that we may have access to particular instances of this function. We use this capability for testing the resulting fusion system. Additionally, the specification of the world can contain theories T_w that capture known dependencies and constraints that the world is known to obey.

Implicit in this formulation is the fact that the goal of the fusion system is to recognize objects in the world, or more precisely, assign object names to all locations in the world. For other goals, the fusion problem could have a different signature, but the idea should remain the same.

The specifications S_1, S_2 represent specifications of two sensors. Each sensor has its own coordinates (X_1 is the coordinate of the sensor specified by S_1 and X_2 is the coordinate of the sensor specified by S_2). Each sensor has its own sort of values (V_1 and V_2). The sensors have their *measurement functions* (f_1, f_2). Moreover, T_1 and T_2 specify theories of sensor operation.

S_c is a collection of simple specs, specifications of *coordinate sorts*. The purpose of identifying these specs is to show the relationships between the world coordinates and the sensor coordinates. They unify sorts that represent the same coordinates. For the example of range and intensity sensors we assume that $X = X_x \times X_y$ and we want to associate X_r with X_x, X_i with X_x and V_r with X_y.

The specification S_f is obtained in two steps. First, a colimit of S_c, S_1, S_2 and S_w is taken. At this point some of the sorts, as explained above, are identified (or "glued" together). This means that some of the sorts listed in the spec S_f would actually be glued and thus that spec would not have as many sorts as shown. For instance, in the case of a range and intensity sensor, three sorts would form an equivalence class $\{X_x, X_r, X_i\}$, and so would $\{X_y, V_r\}$. We left the original sorts names in the specs for the sake of comprehensibility. In the second step the resulting specification is extended by adding the function D_f. Its signature is constructed out of the signatures of the two sensors and of the world. This function takes two measurement functions f_1,f_2 as inputs and returns a decision function that assigns subsets of objects to the world coordinates.

Ideally, the theory T_f of S_f should include the following axiom:

$$T_f \vdash \forall_{x \in X} \ D_f(f_1, f_2)(x) = \Delta(x),$$

which states that the resulting decision function D_f is compatible with the world (specified through function Δ). Such a strong requirement would be very difficult to achieve in practice. This would mean that the fusion system can always find a unique solution and that the decision would always be correct (full agreement with the world). For this reason, we decided not to include this axiom here, but instead, provide a somewhat weaker requirement in the specification S_f.

2.5 Decision Fusion

The solution we have described so far is termed in the literature (cf. [8]) *data fusion.* Another scheme is known as *decision fusion.* In our framework decision fusion is expressed by the diagram of Figure 2.4.

$$\begin{tikzcd}
 & S_d & \\
S_{d1} \arrow[ur] & & S_{d2} \arrow[ul] \\
S_1 \arrow[u] & S_w \arrow[ul] \arrow[ur] \arrow[uu] & S_2 \arrow[u] \\
 & S_c \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.4: Decision Fusion

Here S_{d1}, S_{d2} represent the following specs:

$$S_{d1} = ((X_1, V_1, \Delta : X \to E, f_1 : X_1 \to V_1, D_1 : (X_1 \to V_1) \to (X \to 2^E)), T_{d1})$$

$$S_{d2} = ((X_2, V_2, \Delta : X \to E, f_2 : X_2 \to V_2, D_2 : (X_2 \to V_2) \to (X \to 2^E)), T_{d2})$$

The functions D_1, D_2 are the decision functions for each sensor. They could have been used for making decisions when only one of the sensors is present. In the process of decision fusion these two functions are used instead of raw data. The spec S_d represents the decision fusion block.

$$S_d = ((X_1, V_1, X_2, V_2, \Delta : X \to E, f_1 : X_1 \to V_1, D_1 : (X_1 \to V_1) \to (X \to 2^E), f_2 : X_2 \to V_2, D_2 : (X_2 \to V_2) \to (X \to 2^E), D_d : (X \to 2^E), (X \to 2^E) \to (X \to 2^E)), T_d)$$

Note that in this spec D_d takes the assignments that are the results of application of functions D_1 and D_2 and combines these two assignments into one (fused) assignment.

2.6 Results

In this paper we use our formal framework to show the following:

1. Define the relation *subclass* between two fusion systems.
2. Prove that the class of decision fusion systems is a subclass of data fusion systems.
3. We investigate the issue of when decision fusion is equivalent to data fusion.
   (a) We formulate criteria that need to be satisfied for decision fusion to be equivalent to data fusion.
   (b) We give necessary and sufficient conditions for the equivalence.
   (c) We show examples of equivalent fusion systems.
4. We provide examples of special classes of fusion systems, e.g., monotonic, consistent, preferential and we show that some examples from [8] can be expressed in our framework. For instance, we analyze the example of `AND` fusion, i.e., when $D_d(D_1, D_2)(x) \equiv D_1(x) \cap D_2(x)$.
5. We also discuss how the problem of *data association* is expressed in our framework. This problem is considered to be at the center stage of information fusion.

2.6.1 The "subclass" Relation

In order to be able to compare various fusion systems we introduce the relation of *subclass*, which is a relation between fusion systems.

Definition 2.6.1 *Let S_f^1 and S_f^2 be two data fusion systems like in Figure 2.4, where all nodes except S_f^1 and S_f^2 are the same. We say that S_f^1 is a subclass of S_f^2 if there is a morphism of specifications $\mu : S_f^2 \to \bar{S}_f^1$, where $\bar{S}_f^1$ is a definitional extension of S_f^1, such that the diagrams shown in Figure 2.5 commute.*

$$\begin{tikzcd} \bar{S}_f^1 & S_f^2 \arrow[l] \\ & S_1 \arrow[u] \arrow[ul] \end{tikzcd} \quad \begin{tikzcd} \bar{S}_f^1 & S_f^2 \arrow[l] \\ & S_2 \arrow[u] \arrow[ul] \end{tikzcd} \quad \begin{tikzcd} \bar{S}_f^1 & S_f^2 \arrow[l] \\ & S_w \arrow[u] \arrow[ul] \end{tikzcd}$$

Figure 2.5: Commutativity Requirements for Subclass Relations

Definition 2.6.2 *Two data fusion systems, S_f^1 and S_f^2 are equivalent if both S_f^1 is a subclass of S_f^2 and S_f^2 is a subclass of S_f^1.*

For given S_w, S_1, S_2 and S_c the data fusion systems form a category, where the morphisms from S_f^2 to S_f^1 are the morphisms of specifications that make S_f^1 a subclass of S_f^2.

2.6.2 Decision Fusion as a Subclass of Data Fusion

We assume that both in the case of data fusion and decision fusion the ultimate goal of the system is to make decision, i.e., the goal is to develop a decision function that mimics the world's decision function $\Delta : X \rightarrow E$. In the case of decision fusion, the decision process is decomposed into two steps. First we associate a decision function D_i with each of the sensors, i, and then, in the second step, the decisions (i.e., the results of the application of individual decision functions D_i) are input to the second (fused) decision function D_d. In the case of data fusion, as shown in Figure 2.3, there is only one decision function which operates on sensory data, rather than on decisions based upon data from individual sensors.

The idea that decision fusion is a special case of data fusion is captured by the following theorem.

Theorem 2.6.3 *The class of decision function systems, as specified in Figure 2.4, is a subclass of data fusion systems, as specified in Figure 2.3.*

Proof:

Assume we have a decision fusion diagram as in Figure 2.4. We need to produce a data fusion diagram as in Figure 2.3 such that there is a morphism from the diagram of Figure 2.3 to the diagram of Figure 2.4. As a first step we define S_f as a definitional extension $\bar{S}_d$ of S_d by defining a new function $\bar{D}_f : (X_1 \rightarrow V_1, X_2 \rightarrow V_2) \rightarrow 2^E$ where

$$\bar{D}_f \equiv D_d \circ (D_1 \times D_2).$$

This relation is expressed by the diagram as in Figure 2.6. Let's recall that the definitional extension S_d is equipped with an embedding $S_d \rightarrow \bar{S}_d$ which is the identity on all sorts, operations and axioms from S_d.

We define the arrows from $S_i \rightarrow S_f$ $(i = 1, 2)$ as a composition

$$S_i \rightarrow S_f \equiv S_i \rightarrow S_{di} \rightarrow S_d \rightarrow \bar{S}_d$$

We define the arrow $S_w \rightarrow S_f$ as a composition

$$S_w \rightarrow S_f \equiv S_w \rightarrow S_{di} \rightarrow S_d \rightarrow \bar{S}_d$$

It is clear that the new diagram we constructed is a data fusion diagram as in Figure 2.3. Now the identity morphism $S_f = \bar{S}_d \rightarrow \bar{S}_d$ makes S_d a subclass of S_f according to Definition 2.6.1. Therefore the class of decision fusion systems is a subclass of data fusion systems. This concludes the proof.

∎

The fact that the class of decision fusion systems is a subclass of data fusion systems can be represented by the diagram of Figure 2.7.

$$\begin{tikzcd}
(X_1 \to V_1)(X_2 \to V_2) \arrow[rr, "D_1 \times D_2"] \arrow[dr, "\bar{D}"'] & & (X \to 2^E)(X \to 2^E) \arrow[dl, "D_d"] \\
& (X \to 2^E) &
\end{tikzcd}$$

Figure 2.6: Composition

$$\begin{tikzcd}
& S_d & \\
S_{d1} \arrow[ur] & S_f \arrow[u] & S_{d2} \arrow[ul] \\
S_1 \arrow[u] \arrow[ur] & S_w \arrow[u] & S_2 \arrow[u] \arrow[ul] \\
& S_c \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.7: Decision Fusion is a Subclass of Data Fusion

2.6.3 When is decision fusion equivalent to data fusion?

We have already proved that decision fusion is a subclass of data fusion. Now we investigate the class of data fusion systems that are subclass of decision fusion systems. According to Definition 2.6.2, such a subclass would be equivalent to decision fusion systems.

Note that in our framework, the equivalence of decision and data fusion is expressed by the diagram of Figure 2.8. This diagram says that the data fusion function D_f is of the form $D_f = D_d \circ (D_1 \times D_2)$. Therefore we have the following necessary condition.

$$\forall_{f_1,f_2} \forall_{g_1,g_2} f_1, g_1 : X_1 \to V_1, f_2, g_2 : X_2 \to V_2 D_i(f_i) = D_i(g_i) fori = 1,2 \Rightarrow D_f(f_1, f_2) = D_f(g_1, g_2) \tag{2.1}$$

This condition turns out to be sufficient.

Theorem 2.6.4 *A data fusion system S_f is a subclass of a decision fusion system if and only if the condition of Eq. (2.1) is satisfied.*

Proof:

We need to define a decision fusion system S_d and a morphism $\mu : S_d \to \bar{S}_f$ for a definitional extension $\bar{S}_f$ of S_f. We define S_d to be a definitional extension $\bar{S}_f$, where we extend the specification S_f by adding a function $D_d : (X \to 2^E), (X \to 2^E) \to (X \to 2^E), (X \to 2^E)$, where we set

$$D_d(u_1, u_2) = \begin{cases} D_f(f_1, f_2), & \text{if there exists } f_i \text{, s.t., } D_i(f_i) = u_i \\ \emptyset & \text{otherwise} \end{cases}$$

We can take μ as the identity morphism.

$$\begin{tikzcd}
 & S_d \arrow[d] & \\
S_{d1} \arrow[ur] & S_f & S_{d2} \arrow[ul] \\
S_1 \arrow[u] \arrow[ur] & S_w \arrow[u] & S_2 \arrow[u] \arrow[ul] \\
 & S_c \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.8: Equivalence Decision and Data Fusion

2.7 Classification of Fusion Problems

In this section we analyze special kinds of information fusion which occurs when we add some natural axioms.

These axioms have to relate the the set $D_f(f_1, f_2)$ to the sets $D_1(f_1)$ and $D_2(f_2)$.

Two most important types of fused systems we consider are

1. The Consistent Systems

$$D_f(f_1, f_2) \subset D_1(f_1) \cap D_2(f_2)$$

The consistency of the system allows to show that the fused system is more effective than the sensors used separately. It occurs in a lot of real life applications.

2. The Monotonic Systems

$$D_f(f_1, f_2) \supset D_1(f_1) \cap D_2(f_2).$$

The monotonicity assures that the fused system does not contradict its parts when they are in agreement. We cannot make the general claims about the effectiveness of such systems. In fact this property is considered for theoretical purposes.

3. The Preferential Systems.

$$D_f(f_1, f_2) \subset D_1(f_1).$$

The preferential systems first take into account the reading of the sensor S_1 and only in the case of ambiguity look at the reading of S_2.

4. The Alternative Systems.

$$D_f(f_1, f_2) \subset D_1(f_1) \cup D_2(f_2)$$

The alternative system assures that if the sensors agree then the fused system renders the same decision as both of them, so it is at least as reliable as the least reliable sensor in the system.

The most important class of fused systems is the class of consistent ones. Before we formulate the relevant theorem let us discuss the effectiveness of the system. We use the probabilistic approach, but also the fuzzy logic approach would be possible.

For the sensors S_1, S_2 and a fused system S one might extend the specifications S_1, S_2, S by the measures μ_i on the sets $X_i \times E$ and the measure μ on $X \times E$. These sets of measures must be compatible with the diagram defining S, i.e. for a map $X \to X_i$ the measure μ has to be the pull-back of the measure μ_i.

In such situation we can define two subsets of $X \times E$.

$$\Gamma = \{(x, \Delta(x)) \mid x \in X\}$$
$$D_f = \{(x, D_f(f_1, f_2)(x)) \mid x \in X\}.$$

The effectiveness of of the fused system would be measured by the ratio

$$E(S) = \frac{\mu(\Gamma \cap D_f)}{\mu(D_f)}.$$

By definition the effectiveness would be a number between 0 and 1 which equals 1 when $\Delta(x) = D_f(f_1, f_2)(x)$ outside of some set of measure zero.

Theorem 2.7.1 *Let S be a consistent data fusion system, and let us assume that the axiom (??) is satisfied. Then the fused system is at least as effective as any of its parts.*

Proof.

This just follows from the fact that $D_f(f_1, f_2)(x) \subset D_i(f_i)(x)$ for $i = 1, 2$. This implies that $\mu(D_f(f_1, f_2)(x)) \leq \mu(D_i(f_i)(x))$ and, since the numerator of the effectiveness formula is the same for both systems, the conclusion follows.

Remark. Notice that the effectiveness of the consistent system increases with the set $D_1(f_1)(x) \cap D_2(f_2)(x)$ becoming smaller. This means with additional assumption on independence of the conditions imposed by the sensors S_1, S_2 we will get better effectiveness estimates.

2.7.1 Examples.

In this section we show that some of the fusion setups used in Dasarathy's book [8] fit into our scheme. Let us recall that in [8] the author analyzes several scenarios of fusion systems. These scenarios are not based on the specifications of the sensor output, but on general probabilistic approach. There are two kinds of assumptions.

In the first group of cases the author assumes that each sensor S_i renders a decision about the outcome, with the probability p_i of rendering the right decision and probability q_i of getting the wrong decision, with $p_i + q_i = 1$. The author then estimates the probability of the fused system of getting the right decision for different fusion setups.

These types of setups can be incorporated into our setup as follows. We take the set E to be the set $\{0, 1\}$ with two elements. The element 0 will mean that the system makes the right decision, the element 1 means the system makes the wrong decision.

In the second group of cases the author assumes that each sensor S_i renders a decision about the outcome, with the probability p_i of rendering the right decision and probability q_i of getting the wrong decision, and with the probability r_i of getting a non-decision, with $p_i + q_i + r_i = 1$. The author then estimates the probability of the fused system of getting the right decision for different fusion setups.

These types of setups can be incorporated into our framework as follows. We take the set E to be the set $\{0, 1, \emptyset\}$ with three elements. The element 0 will mean that the system makes the right decision, the element 1 means the system makes the wrong decision, and the element $\emptyset$ means there is no decision.

The coordinate sorts X_i and X are just trivial, i.e. consist of one point.

This setup allows then to perform the analysis of the chapters 2 and 3 of [8]. The setups with parallel sensor suits from chapter 2 correspond to the consistent and monotonic systems, the setups for the serial sensor suites correspond to preferential sensor systems. Thus we can conclude that the situations discussed in [8] are contained in our framework.

Also a very interesting effectiveness analysis from [8] fits into the effectiveness analysis performed above. Since the set X is a one point set, we just need to define the measure on E. In both cases, assigning probabilities p_i, q_i (resp. p_i, q_i, r_i) does exactly that.

2.8 Data Association

The first step in solving the above formulated problem is to perform association, as indicated by the diagram in Figure 2.9. This diagram, similarly as all previous diagrams, shows how to unify the distinguished sorts. It also shows how to unify other sorts and operations. However, we still need to associate values of particular sorts. To achieve this goal, we add an additional specification S_h, which is a definitional extension of S_f. The association is done through change of coordinates:

$$x \rightarrow x_1 \equiv x_1(x) \tag{2.2}$$

$$x \rightarrow x_2 \equiv x_1(x) \tag{2.3}$$

The specification S_h contains four additional functions, x_1 and x_2, as described in the above equation, and additionally f_1' and f_2', defined by the following equations:

$$f_1'(x) = f_1(x_1') \tag{2.4}$$

$$f_2'(x) = f_2(x_2') \tag{2.5}$$

For the example of range and intensity sensors we assume that $X = X_x \times X_y$ and we want to associate X_r with X_x, X_i with X_x and V_r with X_y. We achieve this by creating a new spec S_h in which we add new functions x_1, x_2, f_r' and f_i' where f_r', f_i' have the same signatures as f_r, f_r, respectively. For instance,

$$x_1 = x - a \tag{2.6}$$

$$x_2 = x - b \tag{2.7}$$

$$\begin{tikzcd}
 & S_h & \\
 & S_f \arrow[u] & \\
S_1 \arrow[ur] & S_w \arrow[u] & S_2 \arrow[ul] \\
 & S_c \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.9: Association in Data Fusion

$$f'_r(x) = f_r(x_1) \tag{2.8}$$
$$f'_i(y) = f_r(y_1) \tag{2.9}$$

Thus the formalism of specifications provides a natural framework for data association.

2.9 Syntactic vs. Semantic Fusion

In this section we consider the main fusion block, i.e., the Specification Synthesis block. By specifications we mean signatures (languages), theories over the signatures, and classes of models of the theories. We show the first decomposition of the Specification Synthesis block in Figure 2.10.

In some cases we might be given specific models M_i, M_j, instead of classes of models. Depending on which of the above are available, and depending on some other preferences, the process of developing the fused theory and class of models may be arranged on many different ways. For instance, we might first develop a fused theory T_f, and then find a class of models associated with this theory.

One way to achieve the fusion goal is to split the inputs to the Specification Synthesis block (Figure 2.10) between the two tasks, so that purely semantic information (theories) are input into the Theory Construction task and the semantic inputs (models) are input into the Model Construction task. We denote the semantic task by ∇_T, and the semantic task by ∇_M.

To be consistent with the formulation of the fusion problem in Section ??, the diagram represented in Figure 2.11 must commute. This can be described by the following relations.

$$M \models \nabla_T(T_G, T_i, T_j, T_b)$$
$$M = \nabla_M(M_G, M_i, M_j, M_b)$$
$$M_G \models T_G, M_i \models T_i, M_j \models T_j, M_b \models T_b$$

2.9.1 Syntactic Theory Construction

The Theory Construction task can be considered as a goal-driven process that starts with the goal theory T_G. This theory contains a goal sentence G. The intent is to prove that

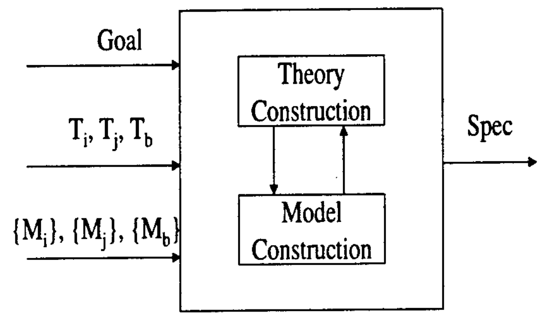

Figure 2.10: Specification Synthesis

$$\begin{tikzcd}
T_G, T_i, T_j, T_h \arrow[r, "\nabla_T"] \arrow[d, "\bullet"'] & T \arrow[d, "\bullet"] \\
\{M_G\}, \{M_i\}, \{M_j\}, \{M_h\} \arrow[r, "\nabla_M"'] & \{M\}
\end{tikzcd}$$

Figure 2.11: Syntactic and Semantic Levels of Specification Synthesis

the goal is true. This theory has to be combined with (extended by) other theories in order to make such a proof possible. It is natural to use the theories of the sensors in the first place. If the goal still cannot be proved, other background theories T_b need to be added. Various standard mathematical theories are added also in this step. The signatures of the goal theory and of the sensor theories, as well as some non-logical symbols appearing in these theories, can be used in the search for theories to add. As a result, we obtain a sufficiently rich theory T (specification) in which all sorts and operations from the goal theory T_G should be definable. In other words, the transition from the goal and sensor theories to the fused theory T can be achieved by appropriate definitional extensions of these theories using the background theories T_b.

2.9.2 Semantic Model Construction

In the Model Construction task we need to combine structures (classes of models of the particular theories fused in the syntactic task) into one class of structures. Since as a net result, this operation should produce such a class of structures $\{M\}$ that each one of them is a model of the fused theory T, the semantic model construction operation ∇_M must be so chosen that this property holds.

2.10 The Fusion Operator

In Section 2.9 we presented fusion as consisting of two operators, ∇_T and ∇_M. What can these operators be? In this section we propose a category theory based approach to this problem, similar to the one taken in the Specware approach [1]. In this approach theories are represented as specifications. They are objects in the Small Categories (*Cat*). Relationships among them are morphisms. Composition of theories is done using the *colimit* operation. Models of the theories are objects of another category (*Mod*).

According to this paradigm, Figure 2.11 can be represented as in Figure 2.12. In this diagram, corners of the diagram represent objects (or collection of objects). The arrows represent morphisms. The operators became:

$$\nabla_T(T_G, T_i, T_j, T_b) = Col(T_G, T_i, T_j, T_b)$$

$$\nabla_T(\{M_G\}, \{M_i\}, \{M_j\}, \{M_b)\} = Lim(\{M_G\}, \{M_i\}, \{M_j\}, \{M_b)\}$$

where *Col* represents the colimit operator and *Lim* represents the limit operator. Note that, since *Lim* and *Col* are two contravariant operators, the morphism arrows point in opposite directions.

According to this diagram, fusion is accomplished by two operators: colimit and limit. The colimit operation combines (glues) two theories (specifications) along the common part. It is a *shared union* of two theories. In other words, first, common parts are identified in the languages associated with particular theories, then these common parts are renamed so that they have the same symbols in both theories, then the renaming is reflected in the axioms of the theories, and finally, the theories are put together into one structure (one theory).

Note also that the arrows from theories to models are of a different kind - they are *functors* that map objects of one category into the objects of another category.

$$\begin{tikzcd}
T_G, T_i, T_j, T_k \arrow[r, "Col"] \arrow[d, "Mod"'] & T \arrow[d, "Mod"] \\
\{M_G\}, \{M_i\}, \{M_j\}, \{M_k\} & \{M\} \arrow[l, "Lim"]
\end{tikzcd}$$

Figure 2.12: Syntactic and Semantic Levels of Specification Synthesis: Category Theory Approach

2.11 Example

2.11.1 Scenario

One of the typical sensor configurations for sensor data fusion is one vision sensor and one range sensor. The vision sensor provides values of (reflected) light intensity for particular sensor coordinates ($X-Y$), while the range sensor provides values of distance from the sensor to the to particular points in the scene represented by the $X-Y$ coordinates. For the sake of simplicity of the presentation we discuss, instead, a simpler and idealized scenario in which we consider only one-dimensional sensor coordinates and ideal (no noise) measurements. One-dimensional sensors, like a line scanning camera and a line scanning range sensor, exist, although as all sensors, they are susceptible to noise, and their measurements are never perfect. We believe that this example is appropriate for the introduction of some of the fusion problems and for discussing solutions to these problems. A model for dealing with noise can be added to this model in a modular fashion.

In this section we describe only some of the components of the scenario. The details will be presented in later sections, after we introduce all the necessary formal concepts.

In our example (see Figure 2.13) we consider a world which is a two dimensional plane whose points are parameterized by two coordinates: x (the ground) and y (the distance from the ground). In this world two kinds of objects are possible: rectangles and triangles (with one right angle). Their sizes vary, but they are relatively small with respect to the size of the world. For simplicity, we assume that one and only one object at a time can be in this world. The world can be empty. Additionally, we presume that objects are stationary, i.e., their position in terms of x and y remains constant in time, and that they are always located between the lines x ($y = 0$) and $l = 1$. The objects are illuminated with parallel light; the light direction is denoted by the angle α, as indicated in the figure. The world is measured through two sensors: a one-dimensional vision sensor, and a one dimensional range sensor. Both sensors make one sweep along line l, as shown in the figure. The vision sensor gives information about the position of each (visible) point of the object with respect to x (the ground) and of the shadow of the object on the ground. The range sensor gives the distance of each (visible) point of the object from line l. Examples of signals from both sensors are

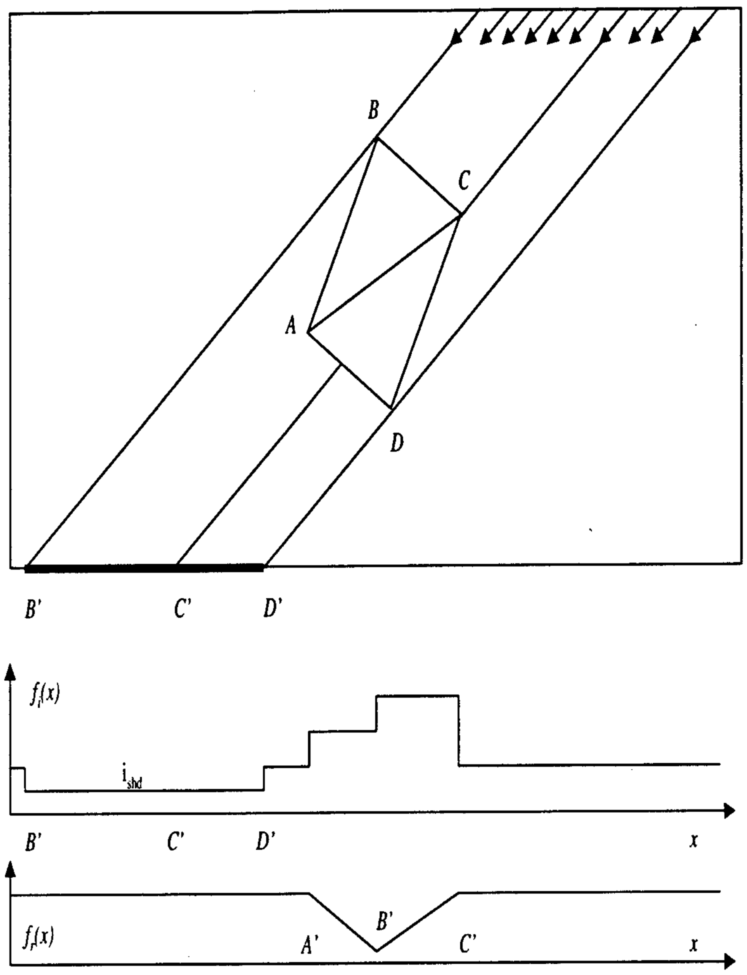

Figure 2.13: A scenario for sensor data fusion

given at the bottom of Figure 2.13.

The goal for the system is to determine whether there is an object in the world and if so, whether it is a rectangle or a triangle with one right angle. Thus the goal is object recognition. Note that in this scenario the range sensor gives more information about the objects than the vision sensor. The range sensor gives not only the location of the object, but also the angles of its sides and the distance from the ground. In some cases the range sensor is sufficient for the classification of an object into one of the three classes. E.g., when an acute angle is at the sensor side, the range sensor gives enough information to classify the object as either a right triangle or as an illegal object. Nevertheless, in some other cases, the information provided by the range sensor is not sufficient to make such a distinction. Yet in some of these cases a less informative vision sensor can come to the rescue. The advantage of the vision sensor stems from its ability to see shadows. In some configurations (sizes of an object and its rotational location), the size of the shadow and its location can provide the extra information that can be used to decide if the object is a triangle or a rectangle. For instance, as shown in Figure 2.13, for both a triangle ABC and a rectangle $ABCD$ the range sensor signal is the same. But the intensity signals are different; for the rectangle the intensity sensor will report shadow for $B'D'$, while for the triangle the shadow will be on $B'C'$. In some other cases, even two sensors do not provide enough information for distinguishing rectangles from triangles.

This example is intended to clearly show how two sensors can provide complementary information about the world and how this information can be utilized by a sensor fusion system. So far we sketched the way this kind of information is fused. To understand this example the reader has to possess some knowledge of geometry and physics. We cannot expect that a computer has this kind of capabilities. Our goal is to understand the mechanisms involved in the above example, formalize these mechanisms, and then implement them in computer. Throughout the rest of this paper we will be using this example to explain all the concepts being introduced.

2.11.2 Formalization of Initial Knowledge

In the following we list the theories involved in the recognition process, show examples of the theories, and describe how they are fused. A complete presentation would include the description of appropriate classes of models. We implemented these theories in the specification language Slang, used by Specware [1], a formal method tool. For readability, however, the theories are presented here using common mathematical notation.

Theory $\mathcal{T}_r$: Range Sensor. The theory of the range sensor, $\mathcal{T}_r$, consists of the following two axioms:

1. $f_r(x) = y \wedge y < 1 \Rightarrow \; O_r(x, y)$
2. $f_r(x) = y \wedge y = 1 \Rightarrow \neg O_r(x, y)$

where 1 is a constant symbol, f_r is a symbol denoting a one-placed function (sensor measurement function), O_r is a symbol denoting a two-placed relation (detection).

Theory $\mathcal{T}_i$: Intensity Sensor. The theory $\mathcal{T}_i$ contains the knowledge about the intensity sensor. It consists of the following single axiom:

$$f_i(x) = i_{shd} \Rightarrow S(x)$$

where i_{shd} - is a constant symbol, f_i is a symbol denoting a one-placed (measurement) function, S is a symbol denoting a one-placed relation (detection of shadow). We have selected a very simple theory of the intensity sensor, since in this example, we use this sensor solely to identify shadows. We extract other relevant information from the range sensor.

Theory $\mathcal{T}_{rt}$: Rectangles and Right Triangles

The theory of rectangles and right triangles contains knowledge that allows us to distinguish rectangles from right triangles. It includes the following predicates: *segment, constant, length, projection, angle, right-angle, acute-angle, triangle, rectangle.* This knowledge is a subset of geometry, and thus is specific to any sensor or to a specific scenario. The axioms of this theory are:

1. $SEG(x_1, y_1, x_2, y_2) \leftrightarrow \forall_{x_1 \le x \le x_2} O(x,y) \wedge y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + y_1 \wedge \forall_{x_2 \le x \le x_1} \neg O(x,y)$
2. $CON(x_1, y_1, x_2, y_2) \leftrightarrow SEG(x_1, y_1, x_2, y_2) \wedge \forall_{x_1 \le x \le x_2} y = y_1 = y_2$
3. $LEN_r(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
4. $PRJ(x, y, \alpha) = x - \frac{y}{tan\alpha}$
5. $SEG(x_1, y_1, x_2, y_2) \wedge SEG(x_2, y_2, x_3, y_3) \rightarrow ANG(x_1, y_1, x_2, y_2, x_3, y_3)$
6. $RAN(x_1, y1, x_2, y_2, x_3, y_3) \leftrightarrow ANG(x_1, y_1, x_2, y_2, x_3, y_3)$
   $\wedge LEN^2(x_1, y_1, x_2, y_2) + LEN^2(x_2, y_2, x_3, y_3) = LEN^2(x_1, y1, x_3, y_3)$
7. $AAN(x_1, y_1, x_2, y_2, x_3, y_3) \leftrightarrow ANG(x_1, y_1, x_2, y_2, x_3, y_3)$
   $\wedge LEN^2(x_1, y_1, x_2, y_2) + LEN^2(x_2, y_2, x_3, y_3) > LEN^2(x_1, y1, x_3, y_3)$
8. $AAN(x_1, y_1, x_2, y_2, x_3, y_3) \rightarrow TRN(x_1, y_1, x_2, y_2, x_3, y_3)$,
9. $RAN(x_1, y_1, x_2, y_2, x_3, y_3) \rightarrow TRN(x_1, y_1, x_2, y_2, x_3, y_3) \vee REC(x_1, y_1, x_2, y_2, x_3, y_3)$.

Theory $\mathcal{T}_{sh}$: Shadows. This theory contains two axioms:

$$SHD(x_1, x_2) \Leftrightarrow \forall_{x_1 \le x \le x_2} S(x) \wedge \forall_{x_2 \le x \le x_1} \neg S(x)$$

$$\begin{aligned} TRN(x_1, y_1, x_2, y_2, x_3, y_3) \wedge \\ RAN(x_1, y_1, x_2, y_2, x_3, y_3) \wedge \\ PRJ(x_2, y_2) = x_l \wedge PRJ(x_3, y_3) = x_r \wedge \\ SHD(x_l, x_r) \Rightarrow TSH \end{aligned}$$

where SHD is the symbol for a two-placed relation (end points of the shadow), TSH – constant representing "shadow of a triangle", and S is part of the language of the theory $\mathcal{T}_r$. The first axiom states that shadows are continuous, and the second axiom defines conditions for when a shadow can be TSH, it is the shadow of a triangle. The idea behind this axiom is the essence of this fusion problem. It can be understood by analyzing the scenario in Figure 2.13.

Theory $\mathcal{T}_w$: The World. In our example we presume that our world can be in three possible states: either it includes a rectangle, or a right triangle, or is empty.

$$\neg(TRN \wedge REC)$$

$$(TRN \vee REC) \wedge \neg TSH \Rightarrow REC$$

Goal: $\mathcal{G}$ The goal of the system is to find our which of the following four situations is the case in the world: (1) there is only a rectangle in the world ($\neg TRN \wedge REC$), (2) there is only a right triangle in the world ($TRN \wedge \neg REC$), (3) there is either a single rectangle or a single triangle in the world ($\neg TRN \wedge \neg REC$) (4) the world contains no objects ($TRN \vee REC$).

2.11.3 Formalization of Fusion in the Example

The specification of the rectangle/triangle recognition system was developed in Slang, the language used by the formal method tool, Specware. Both Specware and the underlying its implementation category theory are described in [10]. The structure of the resulting specification is shown in Figure 2.14.

The specification was developed in a bottom-up fashion. In the first step we developed the specification XREAL. This is an extension of the theory of real numbers (REAL). REAL is one of the theories that is provided with the library of Specware. We needed some additional functions and thus we needed to extend this theory. The arrow from REAL to XREAL is called *import* in Specware. It is an extension of the category theory concept of *colimit* described in [10].

In the next step, theories of the range sensor, $\mathcal{T}_r$, and of the intensity sensor, $\mathcal{T}_i$, described in Section ??, are encoded in Slang. Both theories need to import XREAL. In the Specware implementation they are called RANGE-SENSOR and INTENSITY-SENSOR, respectively. In a similar manner, the RECT-TRIAN specification is created; it imports RANGE-SENSOR and encodes the axioms of the theory $\mathcal{T}_{rt}$. SHADOW imports INTENSITY-SENSOR and encodes the theory $\mathcal{T}_{sh}$.

The next level specification, RECT-TRIAN-SHADOW, is the main fusion block in this whole specification. Here the two theories, RECT-TRIAN and SHADOW, are "glued" together along the common part - the real numbers. In the diagram of Figure 2.14 this common part is shown as the ONE-SORT specification. The sort defined in this specification serves as the common base that unifies the real numbers from the other two specifications. At the same time all the axioms of the two component specifications are mapped into one set of axioms. Then the sort and the operations of this specification are used to extend the colimit by adding additional axioms specified by the theory $\mathcal{T}_{sh}$.

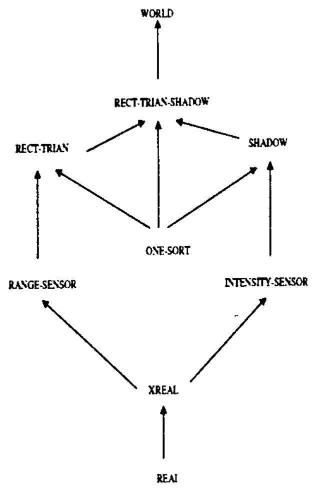

Figure 2.14: Diagram of the Fusion System

2.11.4 Reasoning about the Fusion System

The specification described above can be used for reasoning about the fusion system being specified. For instance, we can reason about the goals of the system. Towards this end, we would have to submit candidate theorems (queries) to a theorem prover and ask whether they could be proven within the theory presented by the specification. In the stage of specification development, such queries could be submitted by either the users (customer side) or by the specification developers (developer side). First, one would need to choose one of the goals from $\mathcal{G}$. The preferable goals are $\{\neg TRN \wedge REC\}$ and $\{TRN \wedge \neg REC\}$, since the success of one of these goals means a precise classification of the object in the scene. The goal $\{\neg TRN \wedge \neg REC\}$ is at the same level of detail. The goal $\{TRN \vee REC\}$ is less specific, since its success means that there is an object in the world, but it is not clear whether it is a rectangle or a triangle. In addition to goals, some information about the inputs, or ranges of inputs, would need to be entered into the system, in order for the prover to resolve the validity of a theorem. The goal is posted to the top level system, WORLD. Since this specification (theory) uses terms from the imported specification, the query is propagated down to that specification, and the process continues until all the truth values can be resolved.

Another application of such a specification is to use it for implementing the system. This can be achieved in the process called *refinement*. In this process, the specification goes through a number of refinement steps (called *interpretations*), the final step being translation into a programming language. Specware supports such a software development process.

Once the system is implemented, its operation can be understood as *model checking* (in logical terminology, (cf. [6]). If the system is implemented according to such a rigorous methodology, as can be easily checked, it will always derive correct decisions, i.e., it will be always right whether the world contains a triangle, a rectangle, one of them, or nothing. The system is not perfect, in the sense that in some situations it will not be able to recognize whether it is a rectangle or a triangle (it will simply say that there is an object in the world: rectangle or triangle). Nevertheless, it can be seen that the fused system will be more powerful than a system with only a range sensor, since it will be able to distinguish between a rectangle and a triangle in all the situations similar to the one shown in Figure 2.13.

2.11.5 Summary of the Example

In this section we provided a formal definition of *fusion* on a small example. Fusion was treated as a formal operator that is applied to two families of objects, *theories* and their *classes of models* and returns a pair – a *fused theory* and a *a class of fused models.* The general fusion procedure consists of two parallel tasks one of the syntactical nature and the second of the semantical nature. Syntactic Theory Construction inputs a goal theory (with a goal formula in it), theories of sensors and background theories and constructs one fused theory for the whole system in which the goal sentence can be proved. Semantic Model Construction inputs models of the theories utilized in the Syntactic Theory Construction task and generates a class of models for the fused theory.

The goal of our research is to find various schemes for performing fusion and to find computationally efficient algorithms to achieve this goal. In this section we showed an example of developing a fused theory, i.e., of the Syntactic Theory Construction. Since we used category theory as our mathematical basis, and Specware as our implementation tool, the correctness of the resulting specification and of the existence of the properties of the specification are guaranteed by the formal semantics of Specware and of the Specware theorem prover.

Formal specifications of fusion systems, like the one described in this section, can serve two purposes. For one, we can reason about various properties of such specifications when we are specifying such systems. This is a very valuable feature, since errors discovered in the specification phase of system development are much cheaper to eliminate than in the later stages. For instance, the same error discovered after deployment of a system can cost hundreds, or even thousands, times more. The other purpose is that such specifications can be transformed into code through the process of refinement. This process guarantees that the specification is implemented correctly. This does not imply, however, that the specification is correct, since this decision depends on the specifier and the user to make. However, having a formally defined specification certainly makes such a process much more reliable and robust.

Chapter 3

Designing Information Fusion Systems

3.1 Towards a Reference Model for Information Fusion

In this chapter we present our research into the design of information fusion systems. Our presentation consists of two parts: a reference model for information fusion systems, and a formal approach to designing information fusion systems. The reference model part was presented in our paper [21]. The formal approach was presented in [12]. Additionally, we have a demo related to this topic which can be found on our project web page (http://www.coe.neu.edu/~kokar).

As was stated in [21], we believe that the problem with designing information fusion systems lies in the way most of us are viewing data fusion. By concentrating on fusion as a means of operating on data, we make the error of thinking in terms of *data fusion systems* rather than*systems with a data fusion capability.*

Research is leveraged through re-use, and in the multi-disciplined field of data fusion re-usability is required, not only of code, but also of architectures, design specifications and operating principles. In our research we take apart conventional notions of fusion process modeling and re-invent it from an object-centered, multiple perspective viewpoint. We do this specifically with the intention of developing re-usable concepts and methodologies for designing and developing a data fusion capability at the system level. From a system design viewpoint, data fusion is not about data but about structures. By combining theories and models rather than data it is possible to reason about the system before the system is actually constructed. We make no claims that we have addressed all of the problems, rather that we have defined a crucial, but neglected, requirement and made the first tentative steps towards its solution.

According to the dictionary definition of the term, to *design* means to (skillfully) work out the structure or form of something, to plan or invent it, as intended for a specific purpose. To work out the structure of a data fusion system and ensure that it is fit for the intended purpose, at least three issues should be addressed:

- A process model should be constructed that is simple enough to allow predictive analysis yet rich enough to admit a variety of complex designs. The model should admit a process of gradual, piece-wise refinement which eventually leads to the deployable system but at the same time should not be overly constraining (and thus giving the designer enough flexibility for choosing design solutions).

- A framework for specifying tasks (goals and solutions) should be developed which permits achievability to be assessed without the need to implement the system. This necessitates a means of representing the state of available knowledge about the world, the information sources and the effectors (but not necessarily a representation of the knowledge itself).
- A theory for the dynamics of the information and of the information processors should be studied. The latter should incorporate both humans and machines and allow for the decision making process to be modeled as a team activity.

In [21] we proposed a fusion reference model that addresses these issues in the following way:

- A prescriptive but tractable object-relationship model
- A formal method for analyzing the relationships between objects
- A mathematical tool for simulating the information flow at the system level
- An appreciation of the psychological factors in the human components

In this chapter we discuss only the first two items with emphasis of the latter one.

3.1.1 Process Model

A process model is a description of a set of processes and the relationships between them. The set of processes such determined should be constructed before the system may be regarded as fully operational. As such it highlights the component functions which the system has but makes no statement regarding their software implementation or physical instantiation. Existing data fusion models (such as JDL3 and Omnibus for example, see Figures 3.1 and 3.2) are still largely descriptive models and thus cover a large number of designs that are it should be able to rule out, even before introducing the constraints imposed by the specific application.

In the fusion model proposed by the US Joint Directors of Laboratories Data Fusion Sub-Group [37], the Dasarathy model [9] and the Waterfall model [30] the data fusion process is divided according to the abstraction of the information being fused. There is therefore no need to make sequencing of functions or processors explicit. Typically the levels of abstraction include:

- Sensor data - such as scalar measurements, waveforms or images
- Signals - which are the result of some elementary signal processing, alignment or registration activities

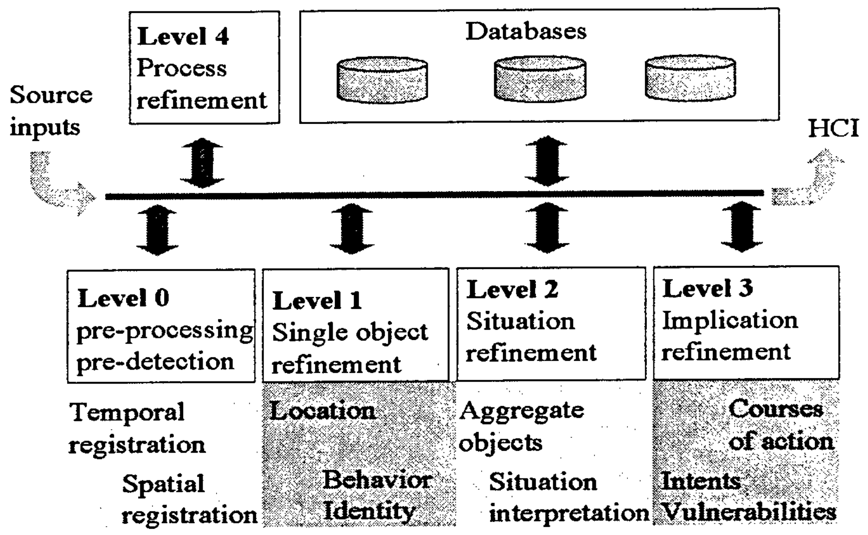

Figure 3.1: The JDL Data Fusion Process Model

- Features - which capture the relevant characteristics of the signal and represents the lowest level at which desired capabilities affect the information being stored
- Object state estimates - which is concerned with the estimation and prediction of continuous (e.g. spatial or kinematic) or discrete (e.g. behavior or identity) states of objects
- Situation estimates - which introduces context by examining the relations among entities, aggregating objects into meta-objects and placing interpretations on the situation
- Planning decisions - implication and possible courses of action are analyzed in light of the current situation

While the models referred to in the previous section are centered on the level of abstraction of the information, the intelligence cycle [35], the Boyd control loop [5] and the Omnibus model [5] are mainly organized from a functional viewpoint. In this case the sequence of functions that are to be performed are made explicit. Each of these models uses a four-stage model:

- Feeding - observing or collecting information and passing it on
- Informing - collating and orientating the information to increase its relevance
- Directing - by evaluating choices and making decisions

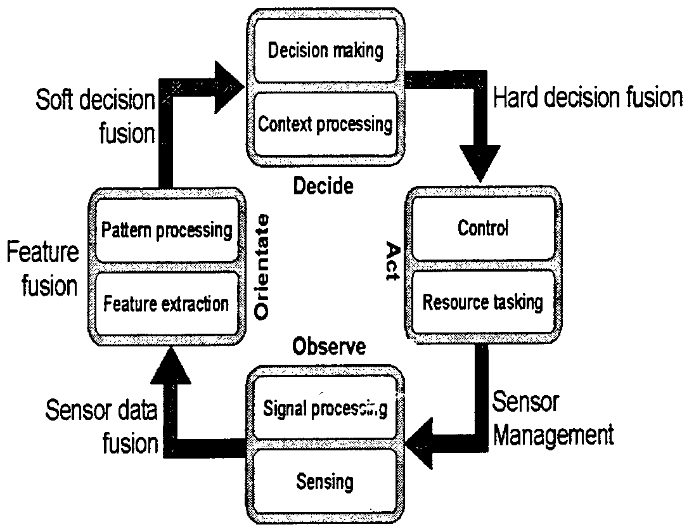

Figure 3.2: The Omnibus Data Fusion Process Model

- Managing - implementing those decisions

Implied by this structure are the agents that perform the functions (actors, perceivers, directors and managers). The merging of these the three viewpoints of information, functions and agents leads us to the object-relationship model presented in the next section.

A natural extension of all of the above models can be captured using a model based on object-oriented design. This is shown in Figure 3.3, where only the first layer class diagram of an object oriented system that implements such a model is shown.

This class diagram shows the agent and information perspectives of the model. The notation is similar to that used in object-oriented design (for instance in UML - the Unified Modeling Language [3]). A class is shown as a box - an actor is an example of a class. A class may be composed of several component classes connected to it via a diamond symbol - for instance a sensor is a component class of an actor.

- Component classes have cardinality constraints - for instance an actor may possess no sensors or many sensors
- A class may have cardinality constraints - for instance an actor must have at least one of effector or sensor
- Two classes may be directionally related by a function - for instance inform is a function of perceiver
- A function has cardinality constraints - for instance each director may be informed by one or more perceivers and zero or more other directors

At the core a fusion system is described as a set of agents (computers or people) that are arranged in the same sequencing loop as the earlier functional models. From the information

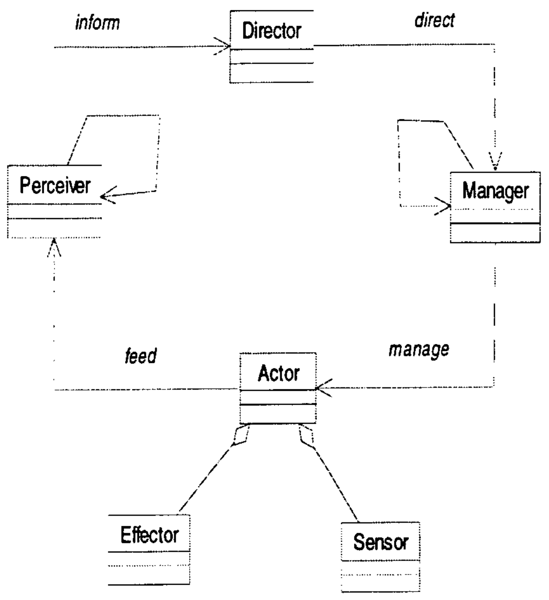

Figure 3.3: First Level Class Diagram of the Fusion Process Model

perspective fusion is described as a set of data abstractions which map directly onto the JDL and other models. The level of abstraction is a consequence of the agents, their ordering and their functions, rather than being imposed as an intrinsic property of the system. By allowing the state class to contain information about this information as well as the environment the loops-within-loops alluded to by Bedworth [5] are easily included. This captures well the ideas that "one man's knowledge is another man's data" (Frank White). Although Figure 3.3 shows only one layer (our paper [21] shows two), this layering of perspectives can continue until the system is deployed (and then further to assist in its maintenance and replacement).

3.1.2 Relationship Analysis

It is important to separate the system knowledge (the data and procedures which operate on that data within the system) from the state of the general knowledge (the availability and quality of data or the specifications of the procedures). The system knowledge will only become available once a version of the system is constructed whereas the state of the general knowledge will be available during the initial design process. This general knowledge will be input to the development process, while the system knowledge will be the output of this process. Consequently, in the development process the knowledge must be manipulated.

As we stated earlier in this report, this feature is a distinguishing one for our approach to fusion: we consider fusion as an automatic (or at least semi-automatic) manipulation of knowledge structures, rather than just data. This point of view raises the level of abstraction - instead of considering algorithms that manipulate data objects, we need algorithms that manipulate algorithms (which in turn manipulate data). As a consequence of this departure point, we need methods that can provide an ability to reason about the performance criteria of fusion systems that are being designed. This kind of ability is typically achieved through the analysis process carried out by the (human) developer of the fusion system (during design) and through simulations (during system verification). The requirement on "reasoning" in the context of the human makes an implicit assumption that the human is responsible for managing the degree of rigor in the reasoning process. The human is expected to carry out a sound reasoning process, rather than just guessing. The human is also expected to relax the rigor, to an acceptable level, whenever there is a lack of information that would be needed for a fully sound process. This kind of a requirement is very difficult to implement in a mechanical reasoner. Typical approaches to this problem are on the edges, either a fully consistent process like logic (but then we encounter the known problems of undecidability) or a heuristic approach like expert systems (but then we cannot guarantee consistency). In order to deal with this problem, we selected a formal method approach based on category theory.

Within the framework of category theory we can address all of the problems stated above. In particular, we can analyze the effects of combining (fusing) structures (e.g.. algorithms) into one structure with respect to the system level criteria. In other words, we can analyze various design choices. The input to this fusion process is the state of the general knowledge (as stated above) and the system level requirements (we also refer to this as "knowledge structures"). The development consists of a number of steps of structure fusion. Since each of the steps is provably correct, i.e., the resulting structure is guaranteed to preserve the system requirements of the previous step, the whole process is provably correct.

The process consists of two phases: specification development and refinement. The specification development phase ends with a complete specification of the system. The refinement phase takes such a complete specification and ends with code. Again, since only formal, provably correct operators are used in the refinement process, the resulting code is provably correct with respect to the specification. In the rest of this chapter, we give the main ideas of our approach.

A *category* [32] is an abstract mathematical construct consisting of category objects and category arrows. Category objects are the objects in the category of interest. In our case they are algebraic specifications of computational objects; the category is called Spec. Category arrows define a mapping from the internal structure of one category object to another and are also called morphisms. In our case they are specification morphisms. In the category Spec, specification morphisms map the sorts and operations of one algebraic specification into the sorts and operations of a second algebraic specification such that the axioms in the first specification become provable theorems in the second specification. Thus, in essence, a specification morphism defines an embedding of one specification into a second specification.

Specification morphisms are required for defining and refining specifications. Additionally, we need the combination, or composition, of existing specifications to create new specifications. This is where category theory is extremely useful in information fusion. Often two specifications that were originally extensions from the same ancestor need to be combined. Therefore, the desired combined specification consists of the unique parts of two specifications and some "shared part" that is common to both specifications (the part defined in the shared ancestor specification). This combining operation is called a colimit [32]. The colimit operation creates a new specification from a set of existing specifications. This new specification has all the sorts and operations of the original set of specifications without duplicating the "shared" sorts and operators.

Diagrams are the main modeling tool used to represent category objects and category arrows. A diagram consists of nodes representing category objects, and arrows, representing category arrows. In our approach, the category objects are the specifications (structures) of sensors, objects of interest (e.g. targets), goals of the system, general knowledge, and system knowledge. A whole system is represented as one diagram. Once such a diagram is developed, one can reason about the properties of the system by using theorem proving. In other words, one needs to state a hypothesis and then prove or disprove the hypothesis within the knowledge included in the diagram.

An example of a diagram for a simple object recognition system was presented in Figure ??. In this figure, the top category object is termed WORLD. It contains specifications of the world constraints, e.g., that two (world) objects cannot appear in the same place at the same time and what are the expected types of objects in this world. This specification embeds the specification called RECT-TRIAN-SHADOW, represented as a morphism between the two category objects. RECT-TRIAN-SHADOW, in turn is formed as a colimit of three specifications below. The lower layer represents specifications of the two sensors involved - RANGE-SENSOR and INTENSITY-SENSOR. At the bottom there are specifications of mathematical objects needed to model the sensors.

As we have shown in our earlier work [25], formal methods (such as category theory) provide a reasoning framework that allows to analyze how abstract, system level performance criteria can affect design choices. Formal methods have recently been successfully applied to the analysis of relatively simple data fusion systems. In order to apply formal methods to design systems for real-word applications of typical complexity, system solutions and theories are needed that constrain the design space. In this paper we outline an approach to combining these methods and tools with other methods and tools that are relevant to the development of information fusion systems.

We have outlined a relationship model for data fusion based on object-oriented principles that goes beyond existing descriptive models and provides a prescriptive model that can be gradually refined to the point where it can be deployed. The model is amenable to analysis at the design stage through the use of formal methods. The information flow in a design may be studied using Petri-net theory. The human aspects of information fusion systems are easily accommodated within the model, since no distinction is drawn between human and computer objects until model is viewed from the realization perspective.

The orthogonality of abstraction and control has not yet been fully integrated into our model. This needs to be resolved before moving on. The design approach also needs to evolve from a largely theoretical model to a practical prescription for efficient system design. The associated tools need to be honed to provide tractable analyses of realistic, full-scale problems. As these analysis tools develop in a fusion setting a network of specifications and theories will need to be established and the fusion object model populated with specifications for generic fusion objects (at least) from the agent and information perspectives. Particular specializations can then inherit the attributes and behavioral characteristics of these objects. The practical experience acquired using a case study has much to offer and we should look towards applying this design philosophy to a medium-scale system in the near term.

Recently, the Unified Modeling Language (UML) has become a de- facto standard for representing software systems. The object diagram shown in this paper does not use the UML notation. It may be useful, for the sake of reaching a wider software engineering community, to translate this diagram into UML. Also, we may incorporate some aspects of the Unified Software Development Process (USDP) to capture the evolution of a fusion system - from specification to implementation. Additionally, in order to combine such a diversity of representations of a fusion system as described in this paper, we may employ the concepts of various views, as promoted by the Reference Model for Open Distributed Processing (RM-ODP), in which five different views of a system are used to represent the interest of various stakeholders.

3.2 FIFF: Formal Information Fusion Framework

In this section we present the results of our efforts to develop a formal framework for developing information fusion systems. In this project we developed elements of such a framework. The most current status of this effort is posted on the project's web page (http://www.ece.neu.edu/groups/ifg/demo.html).

3.2.1 Problem Statement

As we have already said in this report, the state of the art in the area of designing information fusion systems is far from being optimal. While there numerous cases of great solutions for solving various problems where information fusion is required, there is a lack of a methodology for designing systems with information fusion capabilities. In most cases, researchers and engineers propose a solution to a specific problem. In other words, the literature of information fusion is full of solutions that were driven by a specific need, rather than by a need for a design methodology for all kind of information fusion systems. Another problem is that the designer would need to be able to evaluate the solution well before it is implemented. Therefore, we need a framework for such evaluation.

In our research we focus on "generic" information fusion systems. In particular, we are interested in methods for designing of all kind of information fusion systems. We are also interested in techniques that allow the designer to evaluate designs, not just implemented systems. The design knowledge is valuable. Therefore, we would like to be able to reuse such knowledge once it is developed.

In Chapter 2 we showed how we formalize the notion of an information fusion system. We also showed a top-down approach to the development of information fusion systems. In the first step, we identified the "boundaries" of a fusion system (see Figure ??). We identified all inputs and outputs that such a system should have. Then we stated that the system should have a *goal*. The designer then must make sure that the goal is satisfied by the design. The formalization of Chapter 2 includes the formalization of a goal (specification). Then we showed two steps of decomposition of our problem. In the first step we decomposed the life of such a system into three phases: Specification Synthesis, Code Generation and Information Processing (see Figure 2.2. Only in the last phase the system actually performs information fusion. In order to reach this phase, the system first must be developed. We decomposed the development phase into two sub-phases: Specification Synthesis and Code Generation. The phase of specification synthesis requires the expertise of the given domain, i.e., the domain for which the system is developed. For instance, if the system is supposed to recognize airborne targets using various a SAR and an infrared sensor, then there is a clear need for the expertise in the areas of both the targets, including various scenaria, and sensors (in this case radars and IR). The next phase, Code Generation, requires expertise in computer science, more than in the application domain.

In summary, our goal is to develop a Formal Information Fusion Framework (FIFF) with the following characteristics:

- It is generic (applicable to various kind of systems and application domains)
- It is reusable (the designer will be able to reuse specs developed for different systems)
- It has the inference capability (the designer will be able to analyze designs before implementation)
- It guarantees correctness of developed code (it is possible to prove that the code implements a given specification)

Towards this goal we have developed elements of the following:

- Basic vocabulary for specifying and designing fusion systems
- Formal theories of following fusion-related components sensors
  - physical worlds
  - geometry
  - registration (refer to any of means used to make the data from each sensor commensurate in both its spatial and temporal dimensions)
- Mechanisms for organizing, storing, and searching for the above theories using ontology structure
- Basic templates specifying fusion systems
- Procedures for specifying, designing, implementing fusion systems
- Examples/case studies

3.2.2 Use Case: Design an SFS

We started our analysis with a top description of a use case for the activity of designing an Sensor Information Fusion System (SFS). The use case diagram is shown in Figure 3.4. The diagram uses the Unified Modeling Language (UML) [3] notation. The oval represents a use case, i.e., a functionality that is to be included in the system under design. The strawman represents the actor involved in the use case. In this case the actor is the designer of the Information Fusion System. As shown in this figure, the main use case is called "Design SFS". It includes two other use cases - "Define Goal" and "Specify Sensors".

The goal is defined in the language as presented in Chapter 2. The diagram of the goal specification for this case is shown in Figure 3.5.

The specs for this specification are shown in Appendix A.

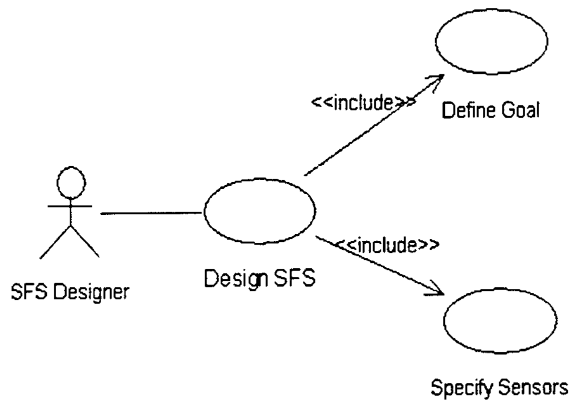

Figure 3.4: Use Case: Design an Information Fusion System

3.2.3 Information Fusion Ontologies

As part of this project we have formalized a number of concepts that are used in the information fusion literature. We expressed these ontologies in Slang, the language of Specware [1]. Two major parts of this ontology are the ontology of geometry and the ontology of sensors. We covered only those parts of the ontologies that were needed for the demonstration purposes. Consequently, only parts of geometry and only a number of sensors have been specified. A hierarchy of sensors in the ontology is shown in Figure 3.6.

The diagram of the geometry ontology is shown in Figure 3.7. The specifications in Slang are shown in Appendix A.

3.2.4 Specification Synthesis

Although the goal specification seems to contain all the necessary information for code generation, it is not quite so. The goal spec contains the signatures of the decision functions. But this spec does not contain enough constraints (axioms) to fully define that function. Therefore, after the goal specification is defined and after the sensor specifications are selected from the ontology repository, the designer has to synthesize the final specification. The final specification must have all the necessary details to uniquely define the decision function. In this process, the designer uses the background knowledge base (the ontology of the background knowledge) and a computer-aided formal method tool, for instance Specware.

The scheme followed by the designer is shown in Figure 3.8. In this figure, KB_s represents the knowledge base (including the sensor and the geometry specifications). K_r represents the subset of KB_s that is *relevant* to the problem specified in the goal spec. K_i represents the subset of K_r that is actually needed to synthesize the final spec. The goal specification is represented as *Problem Spec.* The arrow u stands for the matching function, i.e., the function that determines which of the specs in the ontology are relevant to the given goal spec. The arrow i represents the *insert* operation, or the *import morphism.*

More details of this process are given in the demo on our project's web page (http://www.ece.neu.edu/groups/ifs/demo.html).

3.2.5 FIFF: Features

As we stated earlier in this report, our goal is to develop a framework that has the following characteristics:

1. It is generic.
2. It is reusable.
3. It has reasoning capabilities.
4. It guarantees generation of provably correct code.

To prove the first point, i.e., that the FIFF is generic, we showed that the same procedure can be applied to the design of various fusion systems. This is shown in Figure 3.9. For instance, the framework can be used to design an Automatic Target Recognition system that uses two sensors, a range sensor and an intensity sensor. The same framework can be used to design a surface reconstruction system that reconstructs the surface using three contour sensors.

To show that the FIFF is reusable we need to show that specifications developed for one application can be reused for developing a different system, as shown in Figure 3.10. As shown in this figure, the specs developed for ISF_2, a system for recognizing rectangle and triangles, are used for developing an ATR system that uses a SAR and a MMW radars.

To show the inference capabilities, we need to show that we can formulate and prove theorems about the developed specs. In this step we rely on the theorem proving capabilities of Specware. We used the theorem prover provided with Specware license. Specware has the capability of integrating various theorem provers. For instance, for the rectangle/triangle recognition problem we could formulate a theorem like:

Theorem 3.2.1 $Target(Image(X_1, X_2) = rectangle \Rightarrow Shadow(X_1 - A, X_2 - A)$

This theorem says that if there the system recognizes that there is a rectangle between the X_1 and X_2 coordinates, then there must be a shadow between $X_1 - A$ and $X_2 - A$. In Chapter 4 we show more examples of theorems that can be proved about fusion systems. This capability is very important, since this provides means for rigorous reasoning in the design phase. Otherwise, if we lack a formalization of the problem, the only remaining means are either to make an intuitive assessment or test the performance of the system after it is implemented.

Finally, the last important feature of the FIFF approach is that the code developed using this approach is provably correct with respect to the specification. In other words, the code is guaranteed to satisfy all the requirements of the specification. To show this feature of FIFF all we had to do is to generate code through *refinement*. To show this we did not have to develop any new capabilities, since this capability is within the capabilities of the Specware tool [1].

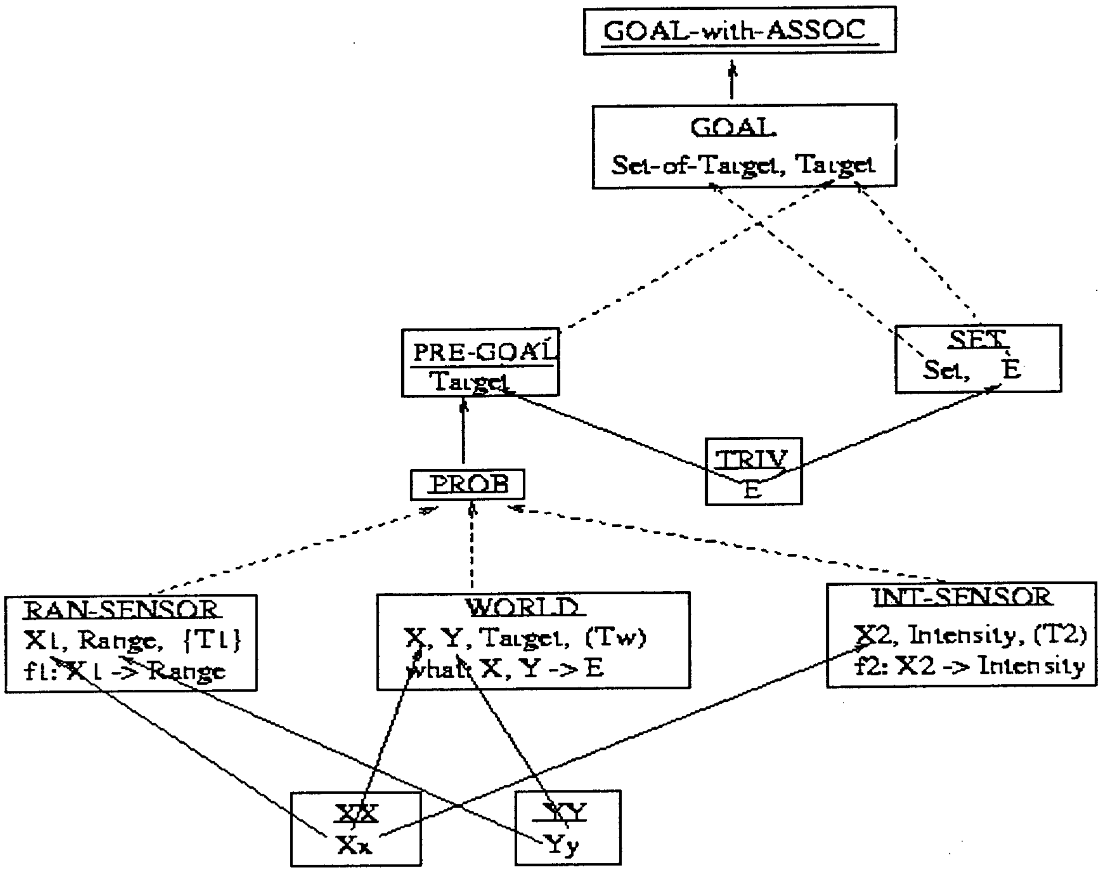

Figure 3.5: Diagram of the Goal Spec for Recognition of Rectangles and Triangles

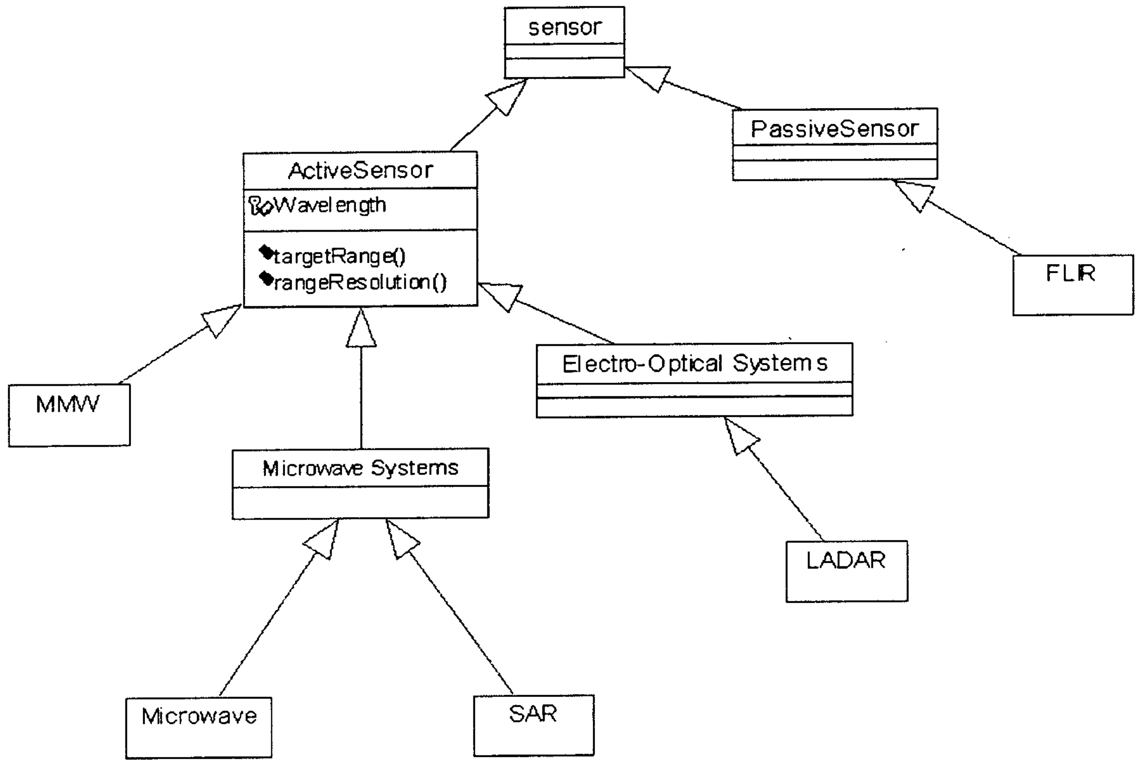

Figure 3.6: Diagram of the Sensors Ontology

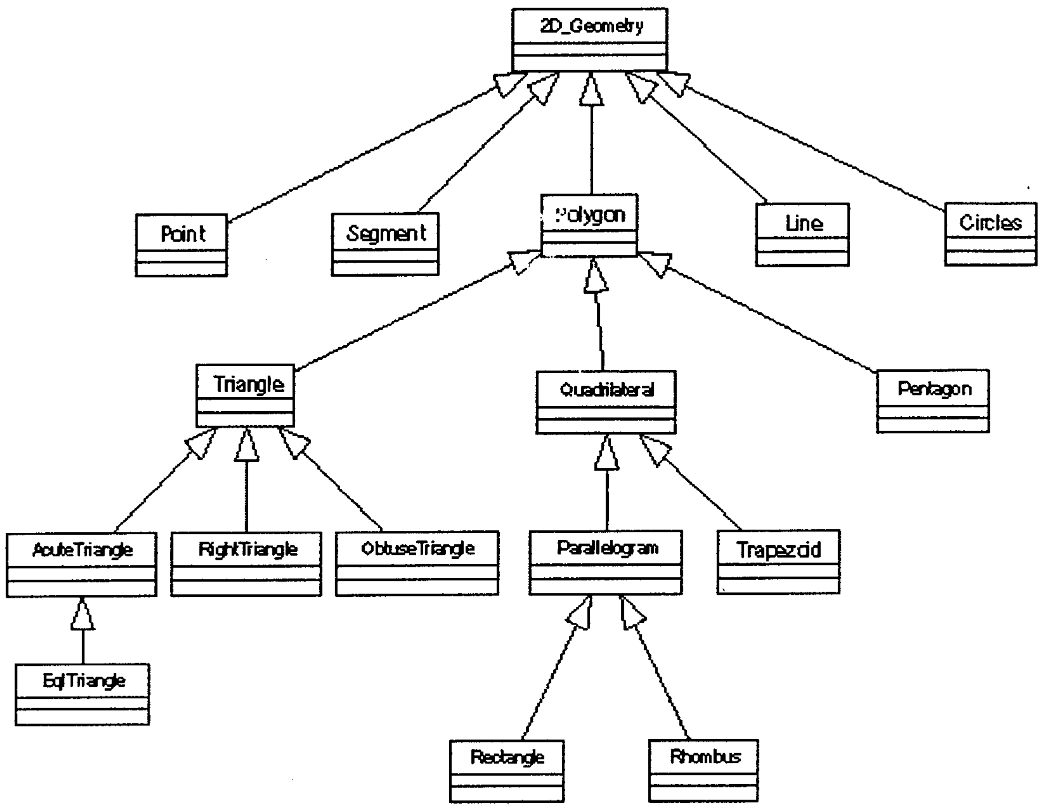

Figure 3.7: Diagram of the Geometry Ontology

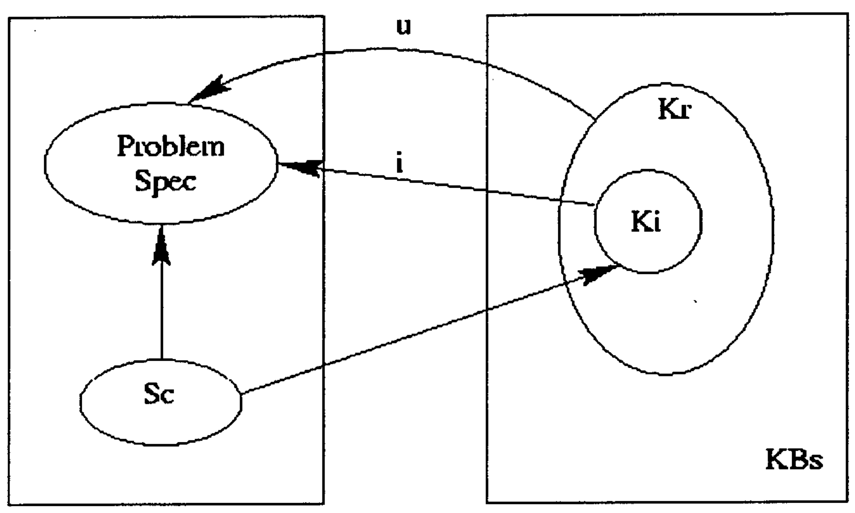

Figure 3.8: Specification Synthesis Scheme

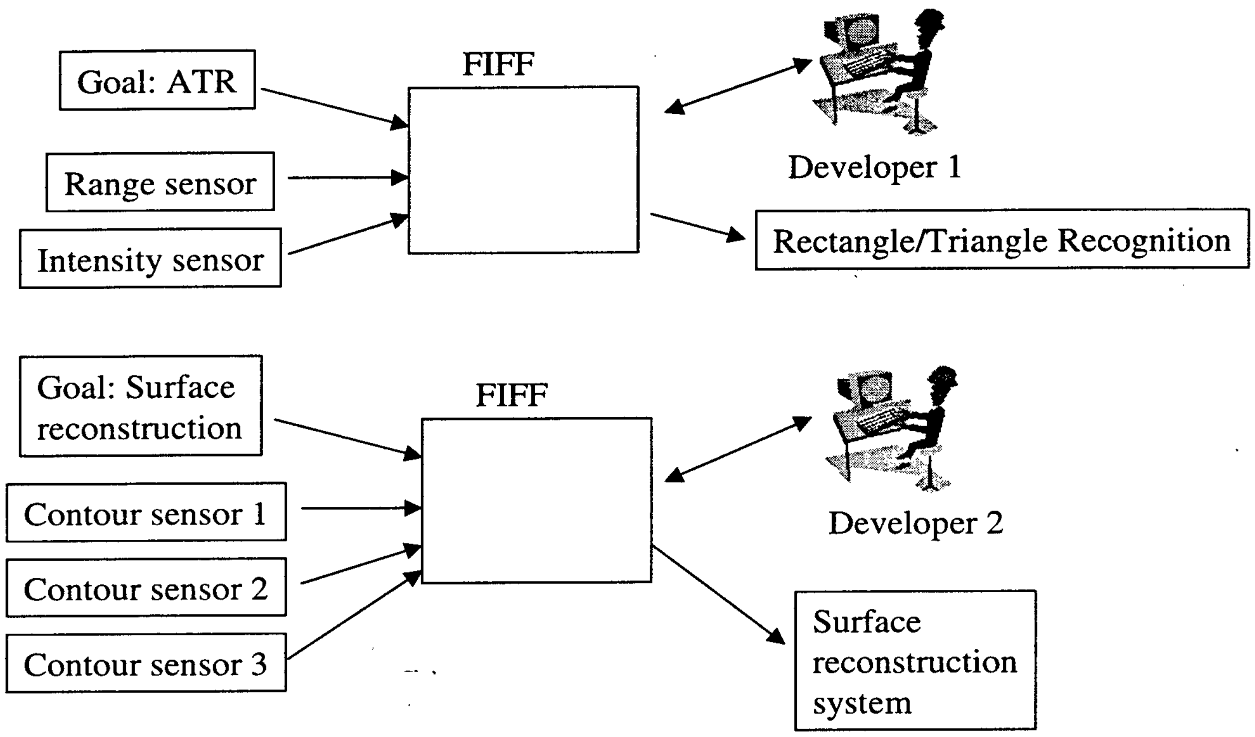

Figure 3.9: FIFF is Generic

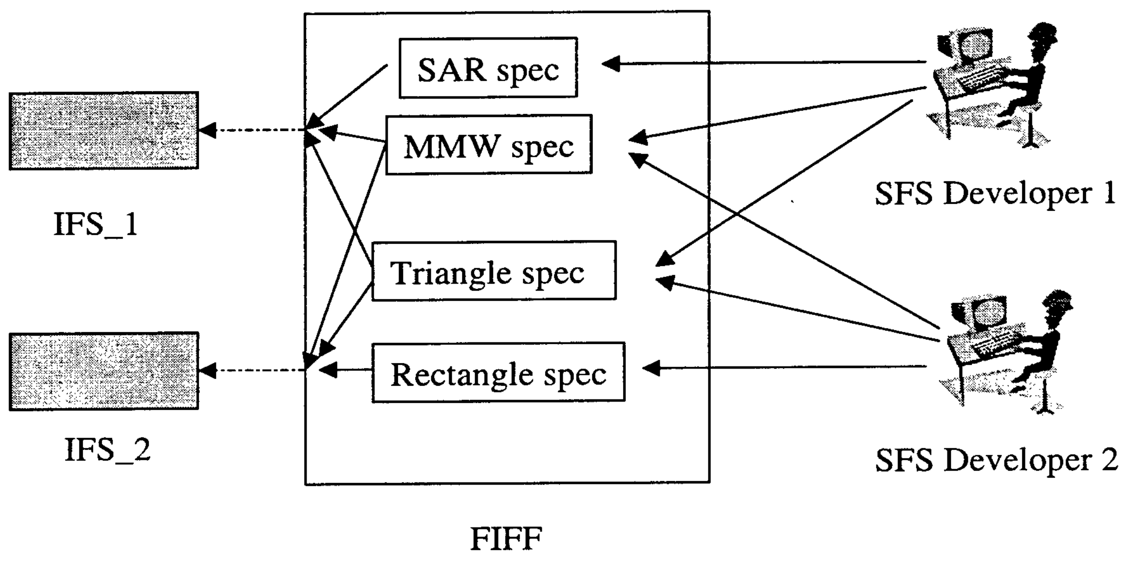

Figure 3.10: FIFF is Reusable

Chapter 4

Reasoning About Uncertainty in the Design Phase

4.1 Introduction

In this section we describe a formal approach to incorporating uncertainty of input information into the fusion process and decision making. Fuzzy set theory (fuzzy numbers, and fuzzy operators) is used to characterize and then manipulate (reason about) uncertainty. A library of specifications of fuzzy set theory is developed using category theory and Specware, a tool that supports category theory based algebraic specification of software. The library is then used to construct specifications of fuzzy information processing systems. The main construction in this process is composition. Category theory operators of limits and colimits are used for composition. As an example, a fuzzy edge detection algorithm is shown, which uses fuzzy operations in its processing. One of the advantages of this approach is that every aspect of the fusion process is specified formally, which allows us to reason about the uncertainty associated with the sensors and the processing.

In information fusion systems, uncertainty of information comes into the picture for a number of reasons: incompleteness of the coverage of the environment, inaccuracy of the sensors (e.g., limited resolution of sensors), background noise in the environment, and others. There are many ways of dealing with uncertainty. Statistical methods and efficient filtering algorithms have been applied to this area using mathematical tools, such as FFT or wavelets, but none in a completely formal way, i.e., these mathematical formalisms have been used to derive algorithms by humans, but not by computing machines (computers).

Why is a formal method so important? We know that in order to design a fusion system, we need to be able to reason about the impact of the uncertainty of the input information on the outcome of the fusion system before the system is built. In other words, we need to be able to predict the performance of the fusion system for any given level of uncertainty and guarantee that it will give satisfactory solutions provided that the uncertainty of incoming information is within some prespecified bounds. With conventional methods, reasoning about the performance of the system cannot be done automatically, but even humans might draw different conclusions about a specific system due to the lack of full mathematical specification of the system.

In this paper, we describe the process by which uncertainty is formally incorporated into the fusion system design, so that it allows us to reason about the uncertainty of the decisions of the fusion system while in the design phase. Section 4.2 describes how a fuzzy set theory library is built using category theory and Specware, and how the library is used to construct specifications of fuzzy information processing systems. This is the main part of the paper. Section 4.3 describes a simple conventional edge detection algorithm, and then maps this algorithm into a corresponding fuzzy edge detection algorithm in which all the operations are replaced by fuzzy operations. This part serves as an example of the application of our approach to reasoning about the uncertainty in information fusion. Section 4.4 concludes the chapter and gives directions for future research.

4.2 Fuzzy Information Processing

Before fuzzy set theory was introduced by Zadeh in 1965, uncertainty was solely treated by probability theory. But there are some situations where uncertainty is non-probabilistic [42]. In information processing systems, for instance, we cannot guarantee that the input data are precise numbers; instead they are often referred to as *approximately x*, or *around x*. The reason for this uncertainty is not that we measure the values with some error, but simply because we do not know what it should be. This uncertainty of imprecision can be modeled by using fuzzy set theory. Another example is evident in linguistic expressions, such as *tall, big, hot,* or *likely, unlikely,* etc. This linguistic uncertainty, of *vagueness* or *fuzziness*, can be well described by appropriate fuzzy sets.

In this paper we use fuzzy set theory to handle uncertainty in information processing systems. We show how fuzzy information processing systems can be specified by using category theory [32] and Specware [1, 2]. Category theory is a mathematical technique that is suitable for representing relations between various types of objects [32]. Specifically, we are interested in relations between (algebraic) specifications. Specware is a tool that supports category theory based algebraic specifications of software [36, 2]. This section will talk about the construction of a fuzzy set theory library and fuzzy information processing specifications.

4.2.1 Construction of Fuzzy Set Theory Library

The fuzzy set theory library is composed of specifications (also called *specs*) of the main concepts of fuzzy set theory: *fuzzy sets, fuzzy numbers, α-cuts*, and *fuzzy arithmetic operations* (cf. [18, 19, 20]. These specs are useful in composing formal specifications of fuzzy information processing systems.

4.2.2 Fuzzy Sets

There are a number of definitions for fuzzy sets. Two most popularly used definitions are listed here for comparison, out of which we chose the second one.

Definition 1 [16]: Fuzzy set A is a set of ordered pairs

$$A = \{(x, \mu_A(x)) | x \in X\}$$

where X is a collection of objects (called *universe of discourse*), and $\mu_A(x)$ is the *membership function*. This function takes real values between 0 and 1.

Definition 2 [20]: Fuzzy set A is a function

$$A : X \to [0,1],$$

where X is the universe of discourse.

The difference between the two definitions is that the former defines a function that is not necessarily total on X, while the latter requires that the function be total. Since Specware requires that all functions be total, we chose the second definition of fuzzy set for building specifications. The diagram of the specification of fuzzy set is shown in Figure 4.1.

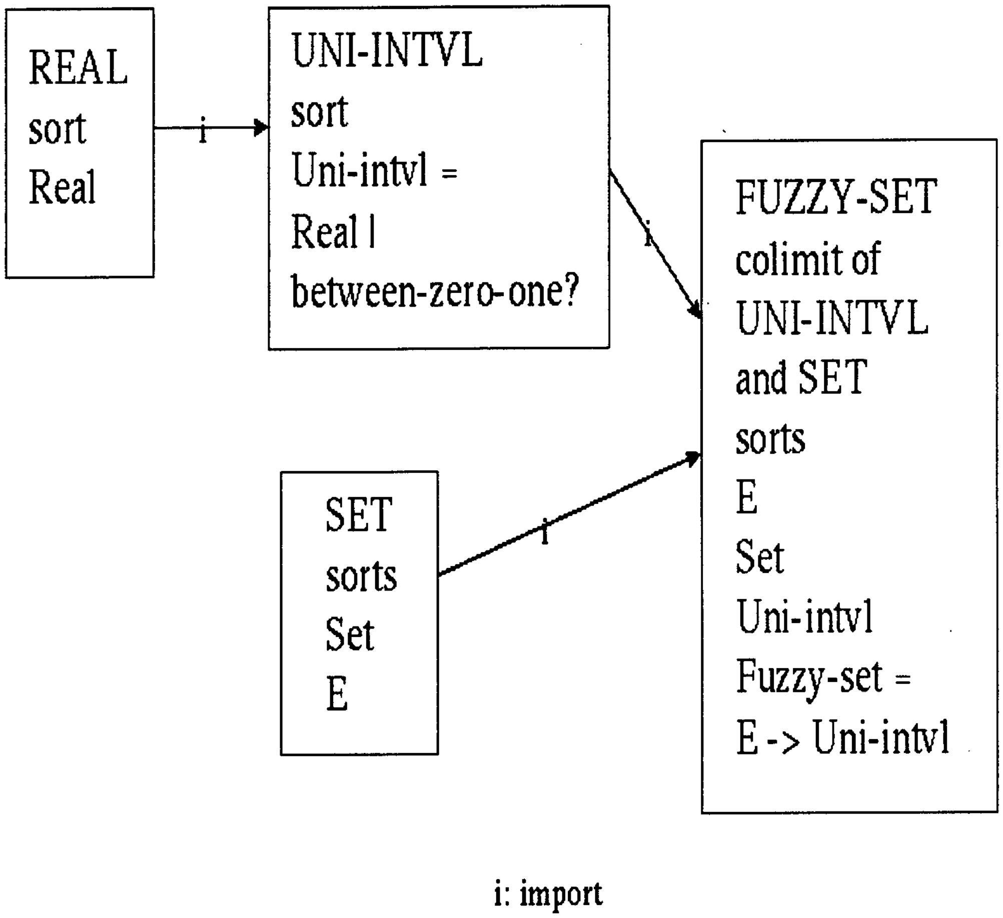

Figure 4.1: Diagram for Fuzzy-set

The spec UNI-INTVL *imports* REAL and introduces a new *sort*: *Uni_intvl* = *Real* | *between_zero_one* FUZZY-SET is a *definitional extension* [5] of the *colimit* of UNI-INTVL and SET; it defines a *function sort*: $Fuzzy\_set = E \longrightarrow Uni\_intvl$, where E is the type of all elements in Set. In the FUZZY-SET spec, α-cut and height are defined as

$$op\ alpha\_cut : Fuzzy\_set, Uni\_intvl \longrightarrow Set$$
$$op\ height : Fuzzy\_set \longrightarrow Uni\_intvl$$

The α-cut is a powerful concepts that links fuzzy sets with sets. The application of the α-cut to a fuzzy set results in a set, and thus all operations and relations of sets can be applied to the α-cuts of the fuzzy set, or to α-levels.

4.2.3 Fuzzy Numbers

Fuzzy numbers are one specific type of fuzzy set. The universe of discourse for fuzzy numbers is real numbers. Fuzzy number A has the form: $A : Real \longrightarrow [0, 1]$. It has the following properties:

- A must be a normal fuzzy set. That is, the height of the fuzzy set A should be 1:

  $$height(A) = \sup_{x \in X} A(x) = 1$$

- A must be a convex fuzzy set. The property of convexity is captured by the following theorem:

  Theorem: A fuzzy set A on *Real* is convex iff

  $$A(\lambda x_1 + (1 - \lambda)x_2) \geq min[A(x_1), A(x_2)]$$

  for all $x_1, x_2 \in Real$ and all $\lambda \in [1, 0]$, where *min* denotes the minimum operator.

- α-cut of the fuzzy set A should be a closed interval for every $\alpha \in (0, 1]$.

These properties are intuitively obvious. A fuzzy number is normal since our concept of a fuzzy number "approximately x" means that it is fully satisfied by x itself. We require that the shape of the fuzzy number be monotonically increasing on the left and monotonically decreasing on the right, so α-cuts of any fuzzy number should be closed intervals, which leads to the property that fuzzy numbers are convex.

Fuzzy number is specified in the spec FUZZY-NUMBER, which *imports* FUZZY-SET and adds one *sort axiom*: $E = Real$. It also adds two axioms: *normality* and *convexity*.

4.2.4 Fuzzy Operations

In [20], two methods have been presented for developing fuzzy arithmetic. One method is based on interval arithmetic. Let A, B denote two fuzzy numbers, * denote any of the four basic arithmetic operations, $+, -, \times, and \div$. Then $A * B$ is a fuzzy number, which can be represented by

$$A * B = \bigcup_{\alpha \in [0,1]} ({}^{\alpha}A *^{\alpha} B) \times \alpha$$

This method requires using α-cuts of fuzzy numbers. The second method represents fuzzy number A*B in the following way:

$$(A * B)(z) = \sup_{z=x*y} min[A(x), B(y)]$$

for all $z \in Real$. We chose the latter one because it is more explicitly expressed, thus more convenient to be specified in Specware.

Fuzzy arithmetic operations are specified in the spec FUZZY-ARITHM which is a *definitional extension* of FUZZY-NUMBER, with fuzzy operations being of the following types.

$$\begin{aligned} op\ f\_add &: Fuzzy\_number, Fuzzy\_number \\ &\quad \rightarrow Fuzzy\_number \\ op\ f\_sub &: Fuzzy\_number, Fuzzy\_number \\ &\quad \rightarrow Fuzzy\_number \\ op\ f\_mult &: Fuzzy\_number, Fuzzy\_number \\ &\quad \rightarrow Fuzzy\_number \\ op\ f\_div &: Fuzzy\_number, Fuzzy\_number \\ &\quad \rightarrow Fuzzy\_number \end{aligned}$$

4.2.5 Fuzzy Information Processing

There are three stages in fuzzy information processing: fuzzification, fuzzy reasoning, and defuzzification. They are covered in the following three subsections.

4.2.6 Fuzzification

The first step in fuzzy information processing is to fuzzify input data. There are many ways to do this. We chose the one in which a triangular membership function is involved. For a given value c, we define the triangular fuzzy number A, such that for all $x \in Real$, $A(x)$ satisfies the equation

$$A(x) = \begin{cases} 0 & \text{if } x < c - \delta, \\ & \text{or } x > c + \delta \\ (x - c + \delta)/\delta & \text{if } c - \delta \leq x \leq c \\ (c + \delta - x)/\delta & \text{if } c \leq x \leq c + \delta \end{cases}$$

In this equation, δ represents the uncertainty level. The larger the δ, the more uncertain the input data.

One kind of typical input data for an information fusion system is image, which is generally sampled into a rectangular array of pixels. Each pixel has an x-y coordinate that corresponds to its location within the image, and an intensity value representing brightness. The spec IMAGE *imports* INTEGER and REAL, and defines a *function sort*: $Image = Integer, Integer \rightarrow Real$. The spec FUZZIFICATION is generated by taking the *colimit* of IMAGE and FUZZY-ARITHM, and defining another *function sort*: $Fuzzy\_image =$

Integer, *Integer* $\rightarrow$ *Fuzzy_number*. The diagram for this specification is shown in Figure 2. FUZZIFICATION maps *Image* to *Fuzzy_image*, so that each pixel has a corresponding fuzzy triangular number instead of a crisp number. Also in this spec, two operations are defined:

$$op\ fuzzify : Real, Nonzero \rightarrow Fuzzy\_number$$
$$op\ fuzzify\_2 : Real \rightarrow Fuzzy\_number$$

where $fuzzify$ takes a crisp number and some uncertainty level, and generates a fuzzy triangular number. The operation $fuzzify\_2$ deals with the situation when the uncertainty level is zero, which means there is no fuzziness about the result. The latter operation is specified so that a crisp number can also be regarded as a fuzzy number.

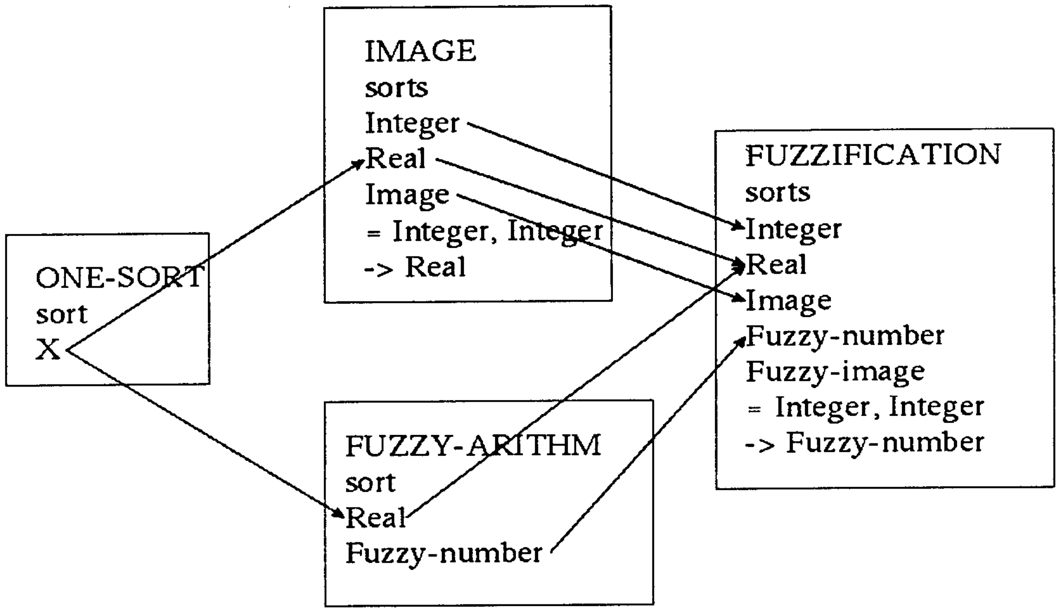

Figure 4.2: Diagram for Fuzzification

4.2.7 Fuzzy reasoning

Fuzzy reasoning takes fuzzified inputs and applies fuzzy arithmetic operations on them. For instance, as we discussed above, the input can be a fuzzy image in which each pixel corresponds to a fuzzy triangular number. While for crisp numbers we apply some arithmetic operations, like $+, -, \times, and \div$, for fuzzy numbers we will apply f_add, f_sub, f_mult, and f_div, as specified in FUZZY-ARITHM. Some additional fuzzy operations are specified there too, which will be useful in our applications. One is fuzzy minimum($fmin$), another is fuzzy maximum($fmax$). Let A, B denote two fuzzy numbers, then

$$\begin{aligned} fmin(A,B)(z) &= \sup_{z=min(x,y)} min[A(x), B(y)] \\ fmax(A,B)(z) &= \sup_{z=max(x,y)} min[A(x), B(y)] \end{aligned}$$

for all $z \in Real$. The results of these two operations are fuzzy numbers. These two operations introduce partial ordering of fuzzy numbers.

Corresponding logic operations such as fuzzy equal($fequal$) and fuzzy less than(flt) are also specified here. There are many ways to define such operations. Here we have chosen the following:

$$\begin{aligned} op\ fequal : Fuzzy\_number, Fuzzy\_number \\ \rightarrow Fuzzy\_number \\ op\ flt : Fuzzy\_number, Fuzzy\_number \\ \rightarrow Fuzzy\_number \end{aligned}$$

The operation $fequal$ takes two fuzzy numbers, defuzzifies them and compares the difference of the result. If the difference is less than a threshold, $fequal$ will return a $fone$, which is generated by $fuzzify(one, \alpha)$. α is the value where the two membership functions intersect and α will be zero if there is no intersection. If the difference is larger than the threshold, $fequal$ will return a $fzero$, which is generated by $fuzzify(zero, \alpha)$. The intersection of the two membership functions are taken to generate α, the same way as in $fuzzify(one, \alpha)$. The result of $fequal$ and flt is either $fone$ or $fzero$. This is the fuzzy equivalent of boolean values true and false. They are not limited to stating whether something is a fact or not, but in addition to this, they give the value of the uncertainty associated with such a statement.

4.2.8 Defuzzification

The input to the defuzzification process is a fuzzy number, and the output is a crisp number. There are several defuzzification methods - *centroid calculation* that returns the center of the area under the curve of the fuzzy number, *middle of maximum* that returns the average of the maximum value of the fuzzy number, *largest of maximum*, and *smallest of maximum*. We chose the *largest of maximum* method to implement the defuzzification process.

Defuzzification is implemented in DEFUZZIFICATION, which is a *definitional extension* of FUZZY-NUMBER. This spec defines the *defuzzify* operation as: $op\ defuzzify$: $Fuzzy\_number \rightarrow Real$. It takes a fuzzy number, finds the largest of maximum of its membership function, and returns the real number as defuzzification result. In our situation we fuzzify the input data using triangular membership function, so after fuzzy operations are applied to these fuzzy triangular numbers, the result will always have only one peak value. Therefore the largest of maximum of its membership function will always return only one value. There are situations where other types of fuzzification are used, and then the defuzzification spec should be more complex.

4.3 An Example: Fuzzy Edge Detection

In this section, we will show how to use fuzzy information processing specifications to translate a standard detection algorithm into a fuzzy detection algorithm, and see how uncertainty of input data propagates during the process and influences the final decision.

4.3.1 Edge Detection Algorithm

An edge in an image could be considered as a boundary at which a significant change of *intensity*, I, occurs. Detecting an edge is very useful in object identification, because edges represent shapes of objects. There are many algorithms for edge detection [4, 26]. The objective of an edge detection algorithm is to locate the regions where the *intensity* is changing rapidly. So we can decompose the whole process into two steps, the first is to derive *edge points* in an image, the second is to apply edge detection method only to these points.

We use the *Laplacian-based* method to derive *edge points*. *Edge points* are where the second-order derivatives of the points are zero, *zero crossing.* So *edge points* can be searched by looking for *zero crossing points* of $\nabla^2 I(x,y)$, which can be calculated by the equation

$$\begin{aligned}\nabla^2 I(x,y) = I(x+1,y) + I(x-1,y) + \\ I(x,y+1) + I(x,y-1) - 4I(x,y)\end{aligned}$$

In order to avoid false edge points, *local variance* is estimated and compared with a threshold. The local variance can be estimated by

$$\begin{aligned}\sigma^2(x,y) = \frac{1}{(2M+1)^2}\sum_{k_1=x-M}^{x+M}\sum_{k_2=y-M}^{y+M}[I(k_1,k_2) \\ -m(k_1,k_2)]^2\end{aligned}$$

where

$$m(x,y) = \frac{1}{(2M+1)^2}\sum_{k_1=x-M}^{x+M}\sum_{k_2=y-M}^{y+M} I(k_1,k_2)$$

with M typically chosen around 2. Since $\sigma^2(x,y)$ is compared with a threshold, the scaling factor $\frac{1}{(2M+1)^2}$ can be eliminated.

The spec EDGE-POINT *imports* IMAGE and defines a *sort* and some *ops*:

$$\begin{aligned}&\textit{sort\_axiom Edge\_point} = \\ &(\textit{Integer}, \textit{Integer})|\textit{edge\_point}? \\ &\textit{op edge\_point}? : \textit{Integer}, \textit{Integer} \\ &\quad \rightarrow \textit{Boolean} \\ &\textit{op grad} : \textit{Integer}, \textit{Integer} \rightarrow \textit{Real} \\ &\textit{op var} : \textit{Integer}, \textit{Integer} \rightarrow \textit{Real}\end{aligned}$$

where *grad* and *var* represent gradient and local variance respectively, and for all Integers x, y:

$$\begin{aligned} &edge\_point?(x,y) \iff \\ &grad(x,y) = 0 \wedge var(x,y) < thrd \end{aligned}$$

Therefore a pixel at (x, y) satisfies an edge point if and only if the gradient equals zero and the local variance is less than the threshold. Otherwise the pixel is not an edge point.

4.3.2 Fuzzy Edge Detection

Now we will use fuzzy information processing specifications and translate the above edge detection algorithm into a fuzzy edge detection algorithm.

Fuzzy edge detection is specified in FUZZY-EDGE-POINT, which *imports* FUZZIFICATION, and defines a *function sort*:

$$\begin{aligned} Fuzzy\_edge\_point = {} & Integer, Integer \\ & \rightarrow Fuzzy\_number \end{aligned}$$

which maps each pixel to a fuzzy number representing the level at which the pixel satisfies an edge point. This fuzzy number represents *fuzzy boolean.* Instead of making the decision that a pixel *is* an edge point or *is not* an edge point, a *fone* or a *fzero* is given. A *fone* states that the pixel satisfies an edge point with uncertainty as described by the fuzziness of this *fone*. A *fzero*, on the other hand, states that the pixel does not satisfy an edge point with uncertainty that is described by the fuzziness of this *fzero*. The following constants and operations are specified:

$$\begin{aligned} const\ delta &: Nonzero \\ const\ thrd &: Real \\ op\ fgrad &: Integer, Integer \rightarrow Fuzzy\_number \\ op\ fvar &: Integer, Integer \rightarrow Fuzzy\_number \end{aligned}$$

where *fgrad* and *fvar* represent fuzzy gradient and fuzzy local variance respectively. Calculation of *fgrad* and *fvar* requires fuzzy arithmetic operations that have been specified before. The operations *fequal*, *flt* and *fmin* are also needed here to realize fuzzy edge detection. The operation *fequal* takes two fuzzy numbers and returns a *fzero* or a *fone*, representing how similar these two fuzzy numbers are. The operation *flt* takes two fuzzy numbers and returns a *fzero* or a *fone*, representing how much the first one is less than the second one. For all Integers x, y:

$$\begin{aligned} & Fuzzy\_edge\_point(x,y) = \\ & fuzzy\_min[fequal(fgrad(x,y), \\ & fuzzify(one, delta), \\ & flt(fvar(x,y), fuzzify(thrd, delta))] \end{aligned}$$

Thus the likelihood that one pixel satisfies an edge point depends on both the likelihood that the fuzzy gradient is close to zero and the likelihood that the fuzzy local variance is less than a threshold. The more the fuzzy gradient is near zero and the fuzzy local variance is far less than the threshold, the more likely this pixel is an edge point. Then $fequal(fgrad(x,y), fuzzify(zero, delta))$ should return a $fone$ with less uncertainty, and $flt(fvar(x,y), fuzzify(thrd, delta))$ should also return a $fone$ with less uncertainty. Therefore $Fuzzy\_edge\_point(x,y)$ corresponds to a $fone$ with less uncertainty.

If $fequal(fgrad(x,y), fuzzify(zero, delta))$ returns a $fzero$, which means fuzzy gradient of the pixel (x,y) is not close to zero with some uncertainty, and if $flt(fvar(x,y), fuzzify(thrd, delta))$ also returns a $fzero$, which means fuzzy local variance of the pixel (x,y) is not less than a fuzzy threshold, then $Fuzzy\_edge\_point(x,y)$ should return a $fzero$, which is the fuzzy minimum of the two results and which shows that the pixel is not an edge point with some uncertainty.

If one of these two operations($fequal$ and flt) returns a $fzero$, and the other returns a $fone$, then $Fuzzy\_edge\_point(x,y)$ should return a $fzero$ which is the fuzzy minimum of the two results. It shows that the pixel is not an edge point with some uncertainty.

4.3.3 Results and Analysis

In order to show that with this approach we can reason about the influence of uncertainty of input information on the final decision before the system is built, we can specify various theorems within a goal spec and then prove them using the Specware theorem prover. As an example consider a GOAL spec, which *imports* FUZZY-EDGE-POINT and introduces a theorem:

Theorem 4.3.1

$$\forall \delta_1, \delta_2 : Unit - Intvl \forall I : Image \forall x, y : Integer$$
$$\delta_1 > \delta_2 \Longrightarrow$$
$$Fuzzy - edge - point(tri - fuzzify - image(I, \delta_1, x, y)) >$$
$$Fuzzy - edge - point(tri - fuzzify - image(I, \delta_2, x, y)),$$

where δ_1 and δ_2 are two different values chosen to fuzzify the input data and represent the uncertainty levels of the input information.

This theorem says that if we take larger values of δ for the step of fuzzification, then the resulting decision about "fuzzy-edgeness" will also be more uncertain (more fuzzy). It is natural that the more uncertain the input data the more uncertain the decision.

In the above example we have applied fuzzy information processing specifications on a standard edge point derivation algorithm and the results show that the uncertainty of input data propagates through the whole process and influences the uncertainty level of the decision. The uncertainty of input data influences the fuzzy gradient and the fuzzy local variance results, which in turn influence the uncertainty of the decision. So instead of giving a crisp decision (true or false), a fuzzy decision is given: true with some uncertainty or false with some uncertainty. The relation between the uncertainty levels in the final decision and in the input information can be proved in this specification stage.

4.4 Conclusions and Future Work

In this chapter we have introduced a formal approach to characterize and manipulate uncertainty in information processing systems. We chose fuzzy set theory to represent uncertainty. We have shown how to specify basic elements of fuzzy set theory in Specware. As an example, fuzzy information processing specifications were applied to an edge detection algorithm. We showed how the uncertainty of input information propagates and influences the final decision

In our project, we have specified a library of the fuzzy set theory by putting in various specifications for fuzzy set concepts. α-cut is a powerful link between fuzzy set and crisp set, so α-cut has been used in our specifications. We have also developed specs of the fuzzy information processing systems. For instance, various fuzzification methods other than triangular have been specified. Trapezoidal, Gaussian, and bell fuzzification methods are three most popularly used. They can represent different levels and kinds of uncertainty among input data or decision making. Fuzzy reasoning includes defining different versions of fuzzy equal and fuzzy less than. Other defuzzification methods have also been specified. All the specifications are listed in Appendix B of this report.

In our future work, we plan to generalize this uncertainty topic by using *random set* instead of *fuzzy set* to characterize and manipulate uncertainty. We also plan to specify random processing and formally introduce *randomness* to some typical information processing problems.

Chapter 5

Wavelet-Based Features for ATR

Wavelets have been used successfully for signal compression. A signal can be represented very concisely and with a high fidelity, by a set of wavelet coefficients. This suggests that wavelet coefficients can efficiently represent the contents of a signal and, consequently, could be used as features. Such features then can be used for signal classification. The quality of classification depends on the choice of the features. Fixing the set of features in both time and frequency domains results in the lack of invariance of the classification method with respect to translations and scaling of signals. In this paper we propose an approach that addresses this problem. We achieve this goal by using the following two techniques. First, our classification method test whether a specific relation among wavelet coefficients is satisfied by a given signal. And second, our method selects features dynamically, i.e., it searches for features that satisfy the relation. The relations are learned from a database of pre-classified signals. In this paper we provide the description of the relation learning approach and results of testing the approach on a simple scenario. The results of our simulations showed that this approach gives a higher classification accuracy than a similar approach based on a fixed set of features.

5.1 Introduction

Wavelets have been used successfully for signal compression. In other words, a signal can be represented very concisely and with a high fidelity, by a set of wavelet coefficients. This suggests that wavelet coefficients can efficiently represent the contents of a signal and, consequently, could be used as features. Such features then can be used for signal classification and recognition. This approach has been investigated by many authors. The problem is, though, which of the wavelet coefficients should be chosen as features? Some of the authors (e.g., Saito [34]) used an entropy based measure to select wavelet coefficients that are most useful as features. This is a fully data-driven approach in which a database of classes of signals is used to select the coefficients that, on the average, have the highest discrimination power. Korona and Kokar [24, 22] used a theory-driven approach, i.e., they assumed that the features were selected by a human expert. Although they have shown that the knowledge-driven approach results in better features, the approach had to rely on the human-in-the-loop. Also, the features generated by both approaches were fixed in both time and frequency domains, i.e., once selected, the same coefficients were used later in the recognition process. In other words, the recognition based upon such features lacks the property of invariance with respect to both time and frequency. This makes both methods inappropriate for scenarios in which signals are that need to be classified as members of the same class are shifted in the time domain or are shifted in the frequency domain.

The main idea of the solution proposed in this paper consists of two parts. One, instead of selecting a fixed set of wavelet coefficients for the whole set of signals, select a most discriminant relation among a number of wavelet coefficients. Two, dynamically attempt to select a set of coefficients such that satisfy a relation for a given class of signals. If such a set of coefficients can be chosen, then the decision is made that the given signal is a member of that class. Otherwise, the decision is negative. A relation is defined as a k-ary relation among wavelet coefficients expressed as a set of k binary relations, where each such binary relation is either "equal", "less than" or "greater than".

The method described in this paper can be used for deriving classification rules for various kinds of signals. In particular, one could use the same approach to generate classification relations for two databases of signals coming from two different sensors. The derived relations could be either used for generating decisions, and then the decisions would need to be fused [8, 40], or the relations could be fused resulting in a single (fused) relation, which then could be used for classification of signals from two different sensors [22].

In the paper we compare the quality of classification by this algorithm with the classification in which a fixed set of the Most Discriminant Wavelet Coefficients are used. Towards this goal, we generate two databases of signals, a training database and a testing database. We show that the misclassfication rate of our algorithm is significantly higher than for the Most Discriminant Wavelets Coefficients.

5.2 Formulation of the Problem

The quality of feature-based recognition strongly depends on the quality of features used. However, there is a trade off between the generality of the algorithm and the quality of recognition. One can always select features that are best for a very narrow type of signals. A "good" set of features, however, is such that is applicable to a wide class of signals, rather than to a very narrow domain.

To formalize the recognition problem we assume the following scenario. First, there is a class of signals. In this paper we consider only 1D signals $s : T \rightarrow V$, where T is time and V is the value set of the signal. We consider features to be collections of pairs

$$\{< t_1, f(t_1) >, \dots < t_n, f(t_n) >\} \tag{5.1}$$

where $t_1, \dots t_n$ are values of the signal domain for a given signal s and $f(t_1), \dots, f(t_n)$ are values of the feature function for the same signal s. We assume that the signals can be classified into classes, say $C_1, \dots, C_K$ by *classification rules.* Often classification rules are represented as relational expressions, e.g.,

$$S_k(< t_1, f(t_1) >, \dots < t_n, f(t_n) >) \Leftrightarrow f(t_1) \geq \tau_1 \wedge \dots \wedge f(t_n) \geq \tau_n \tag{5.2}$$

where S_k $(k = 1, \ldots, K)$ is the classification function for the $k - th$ class and $\tau_1, \ldots, \tau_n$ are thresholds.

In this paper we focus on the selection of features $< t_1, f(t_1) >, \ldots < t_n, f(t_n) >$. In the simplest case these features can be fixed in the time domain, i.e., the values $t_1, \ldots t_n$ are fixed and used by the classification rules. In other words, the values of features are measured (or computed) for the same time instances of the signal's time coordinate. As we mentioned earlier, this is a rather strong constraint, meaning that such a classification rule is not invariant with respect to translations or scaling of the signal. In this paper we investigate a possibility of relaxing such a constraint. Towards this aim, we rewrite the classification rule of Equation 5.2 by replacing the fixed values $t_1, \ldots, t_n$ with the existential quantifier

$$\exists_{t_1,\ldots,t_n} S_k(< t_1, f(t_1) >, \ldots < t_n, f(t_n) >) \Leftrightarrow f(t_1) \geq \tau_1 \wedge \ldots \wedge f(t_n) \geq \tau_n \qquad (5.3)$$

This rule is less restrictive than the rule given by Equation 5.2 in the sense that it can be satisfied whenever there exist the values of $t_1, \ldots, t_n$ such that satisfy the relational expression. The rule of Equation 5.2 on the other hand, requires that this expression needs to be satisfied for the fixed set of values $t_1, \ldots, t_n$. This means that some signals that would not satisfy Equation 5.2, may satisfy Equation 5.3. On the one hand, this may have a negative effect on the quality of classification (since it is easier now to satisfy such a rule). It might be so flexible that all the signals could satisfy a rule and thus the rule is not discriminative enough. But on the other hand, it also gives more flexibility, since this kind of rule can be used for classifying signals that are not exactly the same; signals that have some degree of "similarity" may be assigned to the same class. In this sense, this kind of classification rules can be invariant with respect to both translations and scalings of signals. To achieve the high degree of discrimination power we need to search for appropriate relational expressions; the conjunctive form like in Equation 5.2 might not be sufficient.

This formulation of the classification problem makes it more difficult to formulate classification rules. In the former formulation a classification rule could be "hand-crafted" by selecting a set of thresholds $\tau_1, \ldots, \tau_n$. Various tuning algorithms could be used for this purpose. In the formulation given by Equation 5.3, one needs to find the right hand side of the rule S_k that would replace the simple conjunctive form $f(t_1) \geq \tau_1 \wedge \ldots \wedge f(t_n) \geq \tau_n$. In this paper we present our initial experiments with this kind of problem. In particular, we show experiments with deriving classification rules through training.

5.3 Classification Relations

In this section we describe the type of relational expressions that were used for classification. For finding classification relations we use a Relation Search (RS) algorithm. The relation is selected based upon a training database of signals. In this database, for each signal the signal's classification is given explicitly. The relation selected by the RS algorithm is called Most Discriminant Relation (MDR).

The number of possible relations for a given database of signals is huge. In fact, if we do not fix the arity of the relation, then it is even infinite. In our experiments we limited ourselves to a subset of all possible relations. First, we considered only finite relations of arity $n = 5$ (5-ary relations). More specifically, we focused on sets of five binary relations among wavelet coefficients, where each binary relation was one of the following:

$$B = \{<, =, >\} \tag{5.4}$$

An example of such a 5-ary relation Φ would be

$$\Phi = \{<, <, >, =, =\}. \tag{5.5}$$

We say that an ordered set of features $< t_1, f(t_1) >, \ldots < t_n, f(t_n) >$, where $t_1 < t_2 < t_3 < t_4 < t_5$, satisfies $\Phi = \{r_1, r_2, r_3, r_4, r_5\}$, where $r_1, \ldots, r_5 \in B$, if and only if there exist such values of $t_1, \ldots, t_5$ that $f(t_1) r_1 f(t_2)$ and $f(t_2) r_2 f(t_3)$ and $f(t_3) r_3 f(t_4)$ and $f(t_4) r_4 f(t_5)$ and $f(t_5) r_5 f(t_1)$. It is easy to calculate that there are $3^5 = 243$ possible combinations of such binary relations, i.e., there are 243 possible types of 5-ary relations like this.

The RS algorithm extracts the relation Φ given a set of features for a pre-classified set of signals. The relational formula described above can be used for classifying signals into two classes - those that satisfy the relation and those that don't. For each class of signals such a relation can be constructed giving a method for multi-class classification.

5.3.1 Entropy based classification

The relations described above don't have to be, and usually are not, disjoint. This means that a signal can satisfy multiple relational formulas. In the example considered above, the worst case would be if a signal satisfied all of the 243 formulas. We are interested in such formulas that are most discriminant, i.e., such that give most sharp classifications. Intuitively this means such classifications that have the least of the overlap among the various classes. To measure the quality of a classification rule we use the main idea of the entropy based classification, as for instance presented in a paper by Quinlan [33].

Consider a two-class classification problem, i.e., we have two classes of signals, C_1 and C_2. Let p denote the number of signals in class C_1 and let n denote the number of signals in class C_2. The information associated with this classification is given by Quinlan [33]:

$$I(n, p) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n} \tag{5.6}$$

Now consider a (less than perfect) classification rule Φ that splits the set of signals into two classes. Denote p_1 as the number of signals correctly classified as C_1, n_1 as the number of signals from C_2, but classified as C_1, p_2 as the number of signals correctly classified as C_2, and n_2 as the number of signals from C_1 (incorrectly) classified as C_2. Similarly as in the Quinlan's paper[33], we can calculate the expected information of the classification

$$E(\Phi) = \sum_{i=1}^{2} \frac{p_i + n_i}{p+n} I(n_i, p_i) \tag{5.7}$$

and then the expected information gain due to this classification as

$$gain(\Phi) = I(p, n) - E(\Phi) \tag{5.8}$$

Our relation selection algorithm uses these measures for selecting best relations. It compares information gains for various relations and then selects the one with the highest information gain as the most discriminant relation.

5.4 Experimental Setup

To address the problem described above we performed the experiments as shown in Figure 5.1. First of all, we prepared two databases of signals, a learning database and a testing database. The signals in the two databases have been transformed into the wavelet domain by applying Discrete Wavelet Packet Decomposition (DWPD) [7, 14]. In the next step, the Most Discriminant Bases (MDBs) were selected, one for the learning database and one for the training database, using the Best Discriminant Basis Algorithm (BDBA) as described in the previously referred papers[7, 34]. After that, all the signals were represented in the selected MDBs. At this point we had as many coefficients as samples in each signal. The next step is to select a relation that would be used for classification. Since the goal was to assess our proposed approach against an approach in which a fixed set of features is used, we had to pursue two paths. In one path (the left most branch in Figure 5.1) we applied our relation selection algorithm to all of the coefficients of the MDB (for all signals). In the other path, we selected only five most discriminant elements of the MDB and used the coefficients of these MDB elements in the search for the most discriminant relation. The reason for this was to find out what is the gain in the recognition accuracy that can be attributed to the use of the relational formulation given by Equation 5.3 with respect to the formulation given by Equation 5.2. The result of the two left branches shown in Figure 5.1 were two relations. In the next step, these two relations were applied to classify the test database.

5.5 Simulations

To obtain an initial assessment of the usability of this approach we developed two databases of signals, a training database and a testing database (see Figure 5.1). We generated 100 signals for the learning database and 600 signals for the training database. The signals were of two different classes, rectangular and triangular. In each of the databases there were 50% of signals from each of the two classes. Examples of triangular signals from the learning database are shown in Figure 5.2. As can be seen from this figure, the shapes of the signals were varied. The length of the triangle base, the location of the base in the time coordinate, the location of the apex of the triangle, both the height and the horizontal location with respect to the base, were all varied randomly. Similarly, for the rectangular signals, the size of the rectangle base, its location in the time coordinate, as well as the height, were varied randomly.

One of our goals was to test the sensitivity of this method to noise. Towards this aim, our simulated signals contained various levels of Sensor to Noise Ratio (SNR). We generated signals in such a way so that the distribution of SNR was uniform. The distribution is shown in Figure 5.3.

5.6 Results

To assess the quality of classification we used the measure of *classification accuracy*, δ, defined as a percentage of correct decisions as compared to all the decisions made for the test database.

$$\delta = \frac{P}{P+N} \cdot 100 \tag{5.9}$$

We have run many simulation experiments, each addressed at a specific research question related to the method proposed in this paper. As we said before, our main goal was to assess the usability of the approach proposed in this paper. Towards this aim, we compared the classification accuracy of this method against the accuracy of recognition of a system in which the relations were learned, similarly as in the proposed system, but where the set of features was fixed. As shown in the middle branch of Figure 5.1, we apply the Best Discriminant Basis Algorithm (BDBA) [7, 34] to select a collection of features that are best with respect to the entropy based measure used by the BDBA. Then we select the best relation using our approach. Finally, we classify the signals using both relations, one selected by our approach, and the other that used our approach to the set of features selected by the BDBA. The results of the classification (the classification accuracy) are plotted in Figure 5.4. The dashed line shows the classification accuracy of our method (using the MDR, i.e., most discriminant relations, it is marked as RS Relation in the figure) and the dashed-and-dotted line shows the classification accuracy of the relation among only the most discriminant features selected by the BDBA (in the figure it is called MDB Relation).

5.7 Conclusions and Further Research

Our main goal in this paper was to analyze the usability of a different approach to the classification of signals. This approach uses relations for its decisions, where the relations are learned, rather than defined. The distinguishing characteristic of the formulation of the classification problem described in this paper is the fact that we do not use any fixed set of features, but instead, the systems attempts to find features in the signal that satisfy a given relation. If such a set is found, the decision is positive, i.e., the signal can be classified as being an element of the class associated with the relation. Otherwise, it is a negative decision.

To single out the impact of this approach on the quality of classification we developed two kinds of simulations, one according to our proposed approach, and another one serving as a benchmark. The benchmark simulation also uses relations for decision making, but unlike the approach presented in this paper, it first selects a fixed (such that gives best discrimination power) set of features and then finds a best relation for this set of features. The recognition accuracy based on a fixed set of features does not seem to be a good choice for the scenarios similar to the one presented in this paper due to the fact that the signals in one class were similar in terms of their shape, but were quite different if taken simply as functions of time. For instance, the triangles had their base varying in both size and location, the apex of the triangle varied in both the height and the location with respect to the base. For rectangular signals, the base size and position, as well as the height, were varied.

As can be seen from Figure 5.4, the classification accuracy for the method proposed in this paper is significantly better than for the benchmark approach. It is uniformly better for all levels of SNR tested in our experiments. We attribute this improvement to the use of relations with variable features, as opposed to using fixed sets of features.

While this tells us that the approach proposed in this paper is worth further study, it also requires more work to make it applicable in practice. First of all, the algorithms investigated in this study exhibit a high level of sensitivity to noise. This could be expected since the algorithms did not use any noise filtering or smoothing, but they rather operated on pure signals. This can be improved using standard filtering and smoothing techniques.

Another direction that should be investigated deeper is the choice of the learning and classification algorithms. In this study, we used exhaustive, non-incremental algorithms. In other words, we searched the whole space of 5-ary relations on the whole learning database. There are various learning algorithms that could give better performance in both time efficiency and classification accuracy.

Additionally, we considered only 5-ary relations, where the magic number "5" was selected arbitrarily. This could be a parameter in the relation learning algorithm and thus it would be selected by the algorithm dynamically, rather than fixed at the outset.

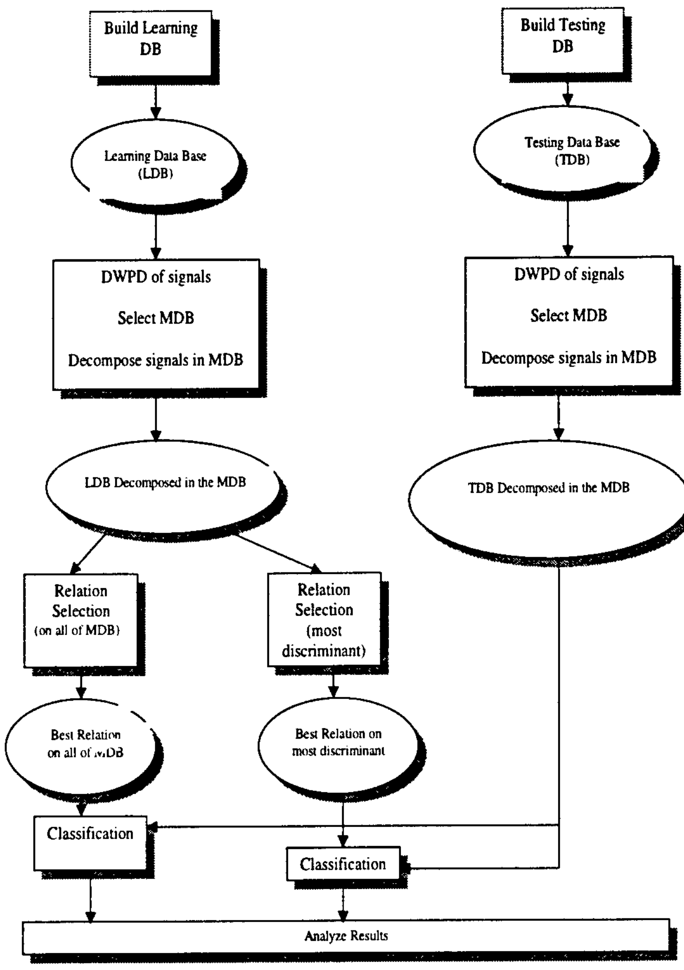

Figure 5.1: Experiments

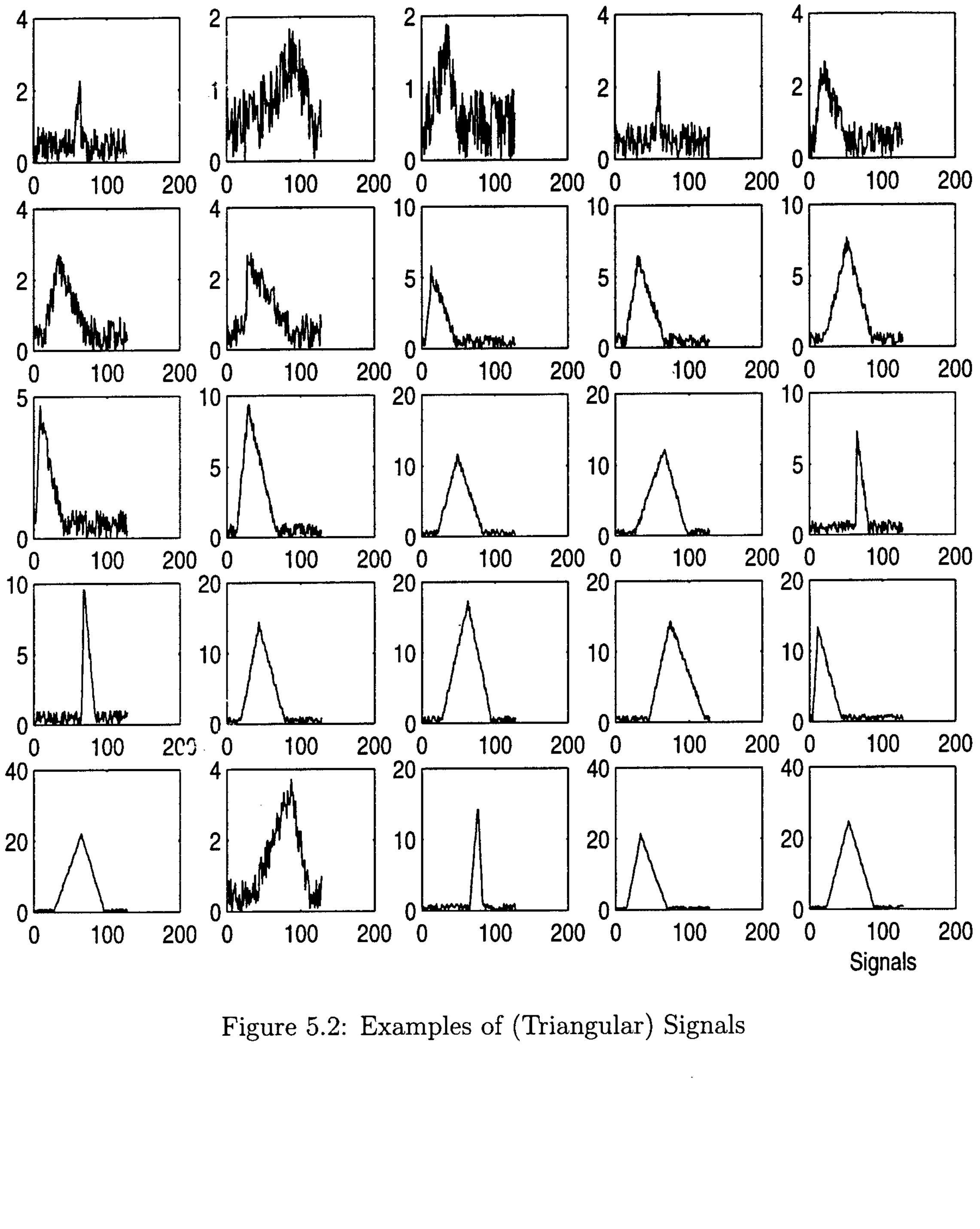

Figure 5.2: Examples of (Triangular) Signals

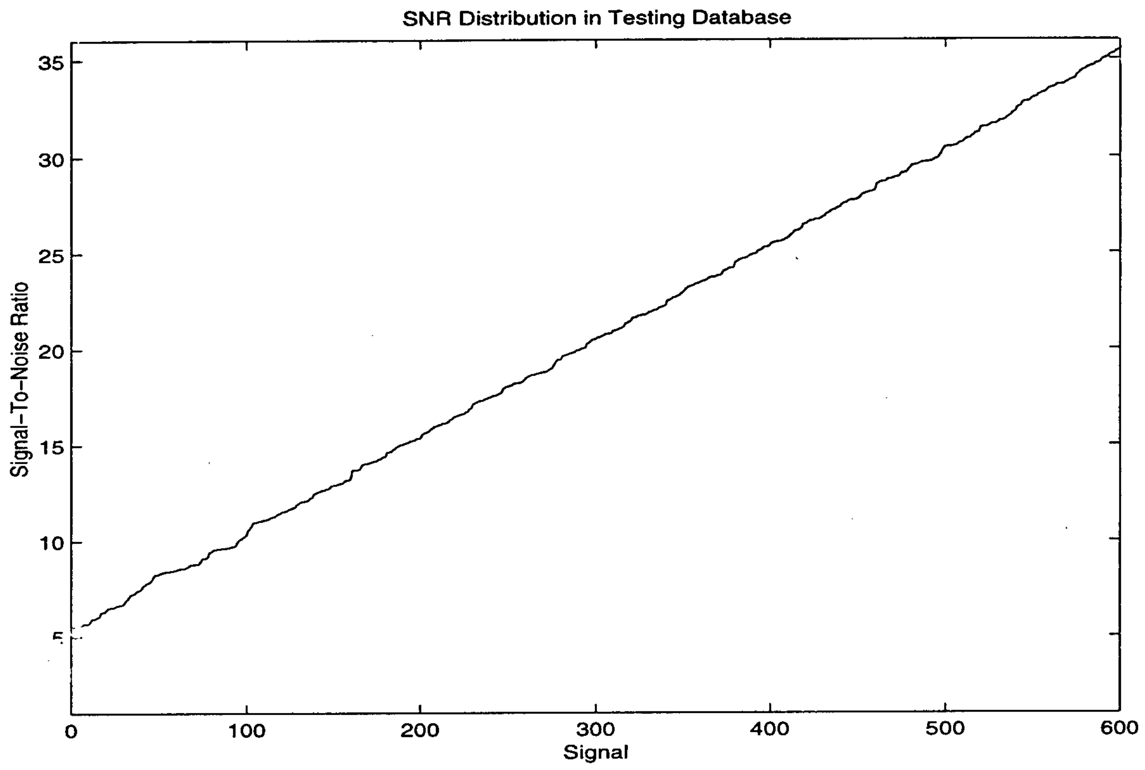

Figure 5.3: Distribution of SNR for Signals in the Learning Database

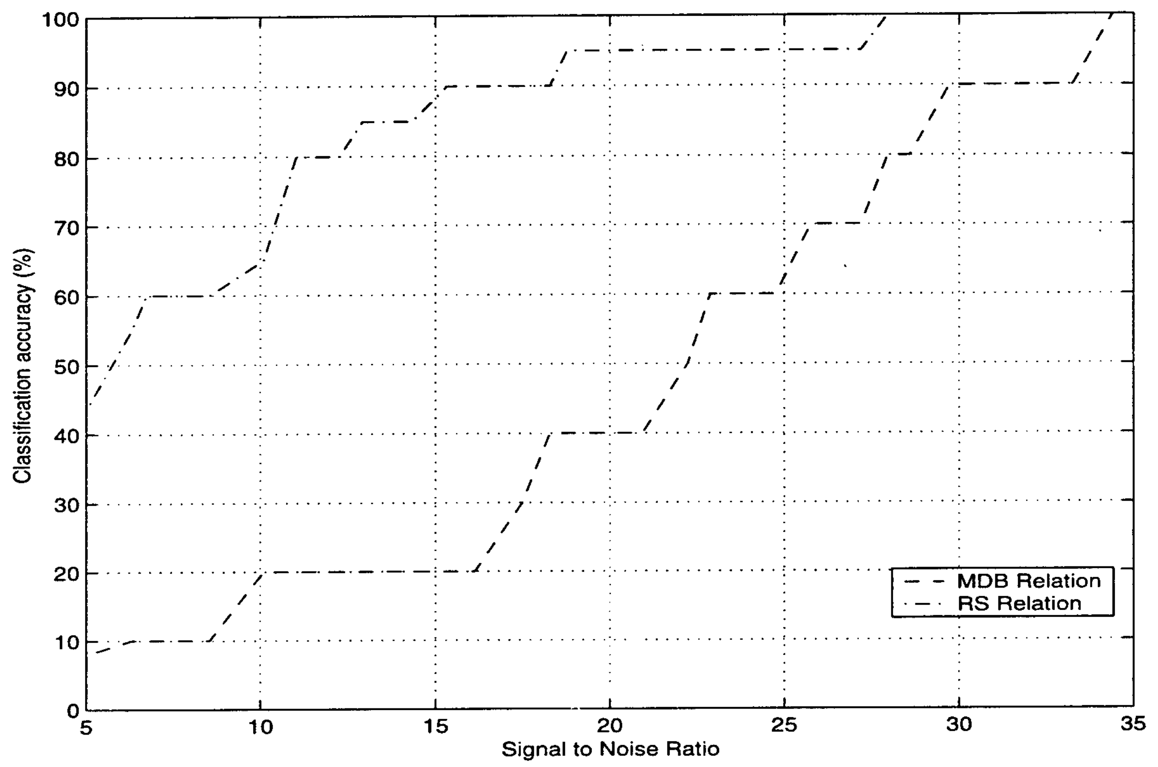

Figure 5.4: Classification Accuracy for Various Levels of Noise (SNR)

Chapter 6

Goal-Driven Autonomous Decision Fusion

6.1 Introduction

This section addresses the issue of decision fusion when two (or more) sensors and the fusion center have a common language to represent queries and decisions, while each of the sensors has its own interpretation of the formulas of the language. Fusion is achieved through the model-theoretic operation of direct product of models. Since not all (most) formulas are not preserved under the product we need an decision procedure that tell us how to combine decisions from particular sensors into one fused decision. Towards this aim the notion of Galvin system is used. The operation of a decision procedure based on this approach is explained on simple examples. The validity of the solution is formally defined and proved in an appropriate theorem. The main advantages of the approach proposed in this section are that the decision mechanism is generic, i.e., it can check the validity of any goal formula, and that it is provably correct.

We consider a case of decision fusion in which all sensors (two or more) derive decisions that are expressed in a language common to all the sensors. Even though it may seem like a very simple case, it is not quite so, because each of the sensors has its own interpretation of the terms of the language. In other words, for each sensor, there is a (different) model associated with the language. Consequently, the process of fusion (cf. [23]) requires that these different interpretations be taken into account when decisions from different sensors are fused.

We address this problem by fusing the interpretation structures (models) rather than just the decisions. In this section we use the operation of *product* to combine structures [6]. Unfortunately, in such a case, even if both sensors derive the same decision, it is not necessarily preserved in the product of two models. For instance, the formula

$$\alpha(x, y) \equiv x \cdot y = 0 \Rightarrow (x = 0 \vee y = 0)$$

most typically does not hold in the product. To be more specific, consider two structures A and B such that $A = B = R$, i.e., both are real numbers with two operations - addition and multiplication under usual interpretation. The formula $\alpha(x, y)$ holds in both A and B, since either x or y must be 0 in order for $x \cdot y$ to be zero. We can say then that $A \models \alpha$ and $B \models \alpha$. In the product $A \times B$, however this is not the case. Note first that in the direct product $A \times B$, the zero element, 0, is represented by the pair $0(0,0)$ and if $x = (x_1, x_2)$ and $y = (y_1, y_2)$ are any elements of $A \times B$ we have $x \cdot y = ((x_1, x_2) \cdot (y_1, Y_2)) = (x_1 \cdot y_1, x_2 \cdot y_2)$. It is easy to see that for $x = (0, 3)$ and $y = (5, 0)$ we have $x \cdot y = (0, 0)4$, but neither x or y are equal 0.

Horn formulas [15], on the other hand, are preserved under products. Even more, Keisler [17] proved that any formula that is preserved under reduced products is equivalent to a Horn formula. Some applications of Horn formulas can be found in [28, 39, 31].

Since, as we stated above, we are interested in fusion by products, our goal in this project is to deal with more general types of formulas, not necessarily Horn formulas, and thus we need a decision procedure which will allow us to decide when a given formula is preserved in the product of two models. Moreover, our objective is to show how such a decision procedure can be derived automatically, i.e., how to construct *autonomous fusion systems* of this sort, given that the system knows the language in which decisions are expressed.

6.2 Problem Formulation

We are addressing here the problem of decision fusion. We assume that the goal of the fusion system is to derive a decision φ based upon decisions $\varphi_1, \ldots, \varphi_n$ obtained from n sensors (n decisions based on inputs from sensors). It is assumed that all sensors have the same language and that they interpret information in structures of the same kind of structure. In our example we assume even more - that carriers of models are the same, although in general it is not important. However, the semantical interpretations of the information can be very different. Our goal is to construct a decision procedure which will assert a formula whenever all the sensors report that some witness formulas holds.

We envision a hierarchical scenario in which there is a central fusion unit that collects inputs from all subordinate units (we call them sensors) and then the central unit makes a decision. The central unit can send various questions (queries) to the sensors. It is possible since both the central unit and the sensors speak the same language.

To better explain the problem we are addressing, we consider the following example scenario (see Figure 6.1). In this scenario, the goal is to recognize whether a detected object (house) has a gable roof oriented in the East-West direction. Two sensors, N and W (north and west) provide reports to the fusion unit. Suppose one of the terms in the language is $GableEW(x)$, which is one of the *goal formulas* of the system. The fusion center, F, can then send the query to the two sensors, W and N. Both sensors will interpret this formula in their own manner. Sensor W will reply "yes" (or "true") when it sees a triangle. Sensor N, on the other hand, will reply "yes" when it sees a rectangle. The fusion center, F, will conclude GableEW(roof) holds if both sensors say so.

Notice that, as we mentioned above, in a general case such a decision procedure is not correct. Our goal in this section is to propose a solution to this problem. More specifically, we will show how to decide about the truthfulness of such formulas. Since the procedure will allow for automatic answering of such queries, we call this procedure an *autonomous fusion system.*

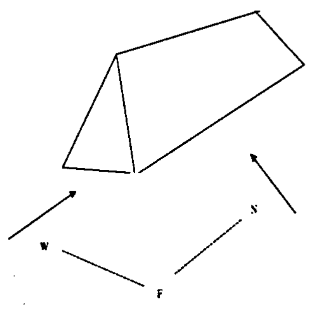

Figure 6.1: Decision Fusion Scenario

6.3 Outline of the Solution

To construct an autonomous fusion system, we use the notion of *Galvin system* (cf. [11, 39]). In the framework of Galvin systems, we show an algorithm for deriving the goal decision φ. In the first step, for any given goal formula φ, our system constructs a set of formulas S and an operation $\Pi : S^n \to S$. The operation Π asserts which of the goal formulas hold in the fused structure (in the product), given that formulas $\varphi_1, \ldots \varphi_n$ hold for each of the sensors. The algorithm then computes a set $T \subset S$, such that the goal formula φ is equivalent to the disjunction of all the formulas from T. In the next step, the algorithm computes $\bar{\varphi} = \Pi(\varphi_1, \ldots, \varphi_n)$ and checks whether this formula is in T. If it is, then the decision φ holds, otherwise it does not. It is clear that since $\bar{\varphi}$ is a disjunct in the set T, $\bar{\varphi}$ implies the goal formula φ.

Popular wisdom has it that when two sensors derive the same decision, then the decision must hold in reality. In this section we show how to rationalize such a rule, i.e., when to accept such a rule in decision fusion and when not to. We have already showed simple examples of decision fusion for both when this popular wisdom rule should be used and when it should not be used. In the following we show how the Galvin system approach works on the scenario of Figure 6.1.

First of all, the fusion center F must select a query, or a *goal formula.* It is obviously related to its higher-level goals. In the next step it analyzes the syntax of the formula. Among others, it identifies *atomic formulas* within the goal formula. (Note, for now we are dealing only with open formulas, i.e., formulas without quantifiers.) In the case of $\varphi \equiv \mathit{GableEW}$, the goal formula consists of one atom. Based upon this analysis, F constructs the set S. In this case $S = \{\mathit{GableEW}, \neg\mathit{GableEW}\}$. In the case of open formulas, S contains all atomic formulas, all conjunctions, and their negations. In the next step, F constructs the mapping Π. In this example it is defined as:

$$\Pi(\varphi, \varphi) = \varphi$$

$$\Pi(\neg\varphi, \varphi) = \Pi(\varphi, \neg\varphi) = \Pi(\neg\varphi, \neg\varphi) = \neg\varphi$$

The set T in this case consists of only one element, $T = \{GableEW\}$. To make a specific decision, the system takes the answers to the query, computes the value of Π and checks whether the result is in T. In this example, the only case that a result is in T is when both sensors say "yes".

6.4 Proof of the Solution

In this section we present a formal definition of the Galvin information fusion system [11, 39] that was informally described above. We also provide a proof that a decision can be reached for any goal formula.

Definition 6.4.1 *A* Galvin Information Fusion System IFS *(with variables $v_1, \ldots v_n$) is a pair (S, Π), where:*

(i) S is a finite set of formulas with variables in $\{v_1, \ldots v_n\}$.

(ii) Π is a commutative and associative operation on S (i.e. (S, Π) is a commutative semigroup).

(iii) For any structure $\mathfrak{A}$ and $a_1, \ldots a_n \in \mathfrak{A}$, there is exactly one formula $\alpha \in S$ such that $\mathfrak{A} \models \alpha[a_1, \ldots a_n]$.

(iv) For any structures $\mathfrak{A}$ and $\mathfrak{B}$ and any elements $a_1, \ldots a_n \in \mathfrak{A}$ and $b_1, \ldots b_n \in \mathfrak{B}$ if for some $\alpha, \beta \in S$ we have $\mathfrak{A} \models \alpha[a_1, \ldots a_n]$ and $\mathfrak{B} \models \beta[b_1, \ldots b_n]$ then $\mathfrak{A} \times \mathfrak{B} \models \Pi(\alpha, \beta)[\langle a_1, b_1\rangle, \ldots \langle a_n, b_n\rangle]$.

Theorem 6.4.2 — *For any goal formula $\varphi = \varphi(x_1 \ldots x_n)$ we can effectively construct a Galvin IFS (S, Π) with variables $x_1, \ldots x_n$ and a set $T \subseteq S$ such that $\bigvee T \leftrightarrow \varphi$ is a tautology.*

Proof: We will proceed by induction. Let φ be an open formula and $\varphi_1, \ldots, \varphi_k$ be all atomic formulas occurring in φ. Let S consists of $\mathbf{2}^k$ formulas of the form $\psi_1 \wedge \ldots \wedge \psi_k$, where each ψ_i is either φ_i or $\neg\varphi_i$. Suppose $\alpha = \psi_1' \wedge \ldots \wedge \psi_k'$, $\beta = \psi'' \wedge \ldots \wedge \psi_k''$. Take $\Pi(\alpha, \beta) = \psi_1 \wedge \ldots \wedge \psi_k$ where $\psi_i = \varphi_i$ if $\psi_i' = \psi_i'' = \varphi_i$ or $\psi_i = \neg\varphi_i$ otherwise. Now Π satisfies all of our requirements and it is easy to see that (S, Π) is then a Galvin IFS and we can find the set $T \subseteq S$ such that $\bigvee T \leftrightarrow \varphi$ is a tautology.

Let us remark now that, if (S, Π) and T satisfy our theorem for φ, then (S, Π) and $S - T$ satisfy our theorem for $\neg\varphi$.

Let $\langle S_1, \Pi_1\rangle$ and T_1 satisfy the theorem for φ_1 and let $\langle S_2, \Pi_2\rangle$ and T_2 do the same for φ_2. Let us define $S = \{\alpha \wedge \beta : \alpha \in S_1 \text{ and } \beta \in S_2\}$, $T = \{\alpha \wedge \beta : \alpha \in T_1 \text{ or } \beta \in T_2\}$, $\Pi(\alpha_1 \wedge \beta_1, \alpha_2 \wedge \beta_2) = \Pi_1(\alpha_1, \alpha_2) \wedge \Pi_2(\beta_1, \beta_2)$. Then it is easy to check that (S, Π) and T satisfy the theorem for $\varphi_1 \vee \varphi_2$.

Finally let $\varphi = \exists x_0 \varphi_1$, and let (S_1, Π_1) and T_1 satisfy the theorem for φ_1. We will find (S, Π) and T for φ. Let $S = \{\alpha_X : X \subseteq S_1\}$ where $\alpha_X = \bigwedge\{\exists x_0 \gamma : \gamma \in X\} \wedge \bigwedge\{\neg\exists x_0 \gamma : \gamma \in S_1 - X\}$. Then it is easy to see that S satisfies the conditions (i) and (iii).

Let $\Pi(\alpha_X, \alpha_Y) = \alpha_Z$, where $Z = \{\Pi_1(\gamma, \delta) : \gamma \in X \text{ and } \delta \in Y\}$. Obviously Π satisfies (ii). We will show that Π satisfies (iv).

Indeed, suppose $\mathfrak{A} \models \alpha_X[a_1, \ldots, a_n]$ and $\mathfrak{B} \models \alpha_Y[b_1, \ldots, b_n]$. To prove (iv), we will show that for any $\beta \in S_1$, $\mathfrak{A} \times \mathfrak{B} \models (\exists x_0 \beta)[\langle a_1, b_1\rangle, \ldots \langle a_n, b_n\rangle]$ iff $\beta \in Z$.

Let $\beta \in Z$, then there is $\gamma \in X$ and $\delta \in Y$ such that $\beta = \Pi_1(\gamma, \delta)$. Moreover since $\mathfrak{A} \models \alpha_X[a_1, \ldots, a_n]$, we have $\mathfrak{A} \models (\exists x_0 \delta)[a_1, \ldots, a_n]$ and in the same way $\mathfrak{B} \models (\exists x_0 \delta)[b_1, \ldots, b_n]$. Thus there are $a_0 \in A$ and $b_0 \in B$ such that $\mathfrak{A} \models \delta[a_0, \ldots, a_n]$ and $\mathfrak{B} \models \delta[b_0, \ldots, b_n]$. Consequently $\mathfrak{A} \times \mathfrak{B} \models \Pi_1(\gamma, \delta)[\langle a_0, b_0\rangle, \ldots, \langle a_n, b_n\rangle]$. but $\beta = \Pi_1(\gamma, \delta)$, whence $\mathfrak{A} \times \mathfrak{B} \models \beta[\langle a_0, b_0\rangle, \ldots, \langle a_n, b_n\rangle]$ and $\mathfrak{A} \times \mathfrak{B} \models (\exists x_0 \beta)[\langle a_1, b_1\rangle, \ldots, \langle a_n, b_n\rangle]$.

Conversely, let $\mathfrak{A} \times \mathfrak{B} \models (\exists x_0 \beta)[\langle a_1, b_1\rangle, \ldots, \langle a_n, b_n\rangle]$. Then there is $\langle a_0, b_0\rangle \in A \times B$ such that $\mathfrak{A} \times \mathfrak{B} \models \beta[\langle a_0, b_0\rangle, \ldots, \langle a_n, b_n\rangle]$. Let $\gamma \in S_1$ be such that $\mathfrak{A} \models \gamma[a_0, \ldots a_n]$ and $\delta \in S_1$ be such that $\mathfrak{B} \models \delta[b_1, \ldots, b_n]$. Then $\mathfrak{A} \models (\exists x_0 \gamma)[a_1, \ldots, a_n]$ and by definition of α_X, $\gamma \in X$. In the same way $\delta \in Y$. Moreover $\Pi_1(\gamma, \delta) = \beta$, thus $\beta \in Z$ and Π satisfies (iv).

This completes the proof.

6.5 Conclusions

In this section we addressed the issue of decision fusion. We assumed the situation in which various sensors and a central decision fusion system share a common language. More precisely, the syntax of the language is common to all parties. But the interpretation of particular symbols is different for each sensor. The fusion process is based on the model-theoretic operation of direct product of models. Accordingly, the fusion process takes all the decisions from the particular sensors and then derives a positive decision only if all the sensors agree. In this scenario, the fusion center must be able to send queries to the sensors. This query is dependent on its goal represented by a goal formula. In case of a complex query, the fusion center analyzes the structure of the goal formula and derives a decision derivation procedure. This decision process is based upon the notion of Galvin system. In this section we showed simple examples of how such a system works. We also showed the formal derivation of the correctness of such procedure.

Decision fusion is important in many applications. The scenario in which particular sensors have their own interpretations is quite typical. For instance, a radar "sees" an object in a different way than a vision camera, an infrared camera or an ultrasound. All of these sensors, however, are used in target detection. Decisions from such sensors need to be fused. Typically, fusion algorithms are constructed around a specific set of goal formulas. If an additional formula needs to be added, the system must be redesigned and re-implemented. The approach presented in this section resolves such a problem by proposing a generic decision mechanism that can check the validity of any goal formula. The main advantage of this mechanism is that, unlike heuristic rule-based approaches, it is provably correct.

Bibliography

[1] Specware: Language manual, version 2.0.3. Technical report, Kestrel Institute, 1998.

[2] Specware: User guide, version 2.0.3. Technical report, Kestrel Institute, 1998.

[3] Omg unified modeling language specification, version 1.3. Technical report, Object Management Group, 1999.

[4] G. A. Baxes. *Digital Image Processing.* John Wiley & Sons, Inc., 1994.

[5] M. D. Bedworth and J. C. O'Brien. Pittfalls in data fusion (and how to avoid them). In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 437–444, 1999.

[6] C. C. Chang and H. J. Keisler. *Model Theory.* North Holland, Amsterdam, New York, Oxford, Tokyo, 1992.

[7] R. Coifman, R. and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38, no.2:713–718, 1992.

[8] B. V. Dasarathy. *Decision Fusion.* IEEE Computer Society Press, 1994.

[9] B. V. Dasarathy. Sensor fusion potential exploitation - innovative architectures and illustrative applications. *Proceedings of IEEE*, 85, No.1:24–38, 1997.

[10] S. A. DeLoach and M. M. Kokar. Category theory approach to fusion of wavelet-based features. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 117–124, 1999.

[11] F. Galvin. Horn sentences. *Annals of Mathematical Logic*, 1:389–422, 1970.

[12] H. Gao, M. M. Kokar, and J. Weyman. An approach to automation of fusion using specware. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 109–116, 1999.

[13] D. L. Hall. *Mathematical Techniques in Multisensor Data Fusion.* Artech House, Boston - London, 1992.

[14] L. Hong. Multiresolutional filtering using wavelet transform. *IEEE Transactions on Aerospace and Electronic Systems*, 29(4):1244–1251, 1993.

[15] A. Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16:14–21, 1951.

[16] J. R. Jang and C. Sun. Neuro-fuzzy modeling and control. *IEEE Transactions*, 1995.

[17] H. J. Keisler. Reduced products and horn classes. *Trans. Amer. Math. Soc. Ser.2*, 81:307–328, 1965.

[18] G. J. Klir. On the alleged superiority of probabilistic representation of uncertainty. *IEEE Transactions on Fuzzy Systems*, 2, No.1, 1994.

[19] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prentice Hall PTR, 1995.

[20] G. J. Klir and B. Yuan. *Fuzzy Set Theory Foundations and Applications.* Prentice Hall PTR, 1997.

[21] M. M. Kokar, M. D. Bedworth, and K. B. Frankel. A reference model for data fusion systems. In *Sensor Fusion: Architectures, Algorithms, and Applications IV*, pages 191–202. SPIE, 2000.

[22] M. M. Kokar and Z. Korona. A formal approach to the design of feature-based multi-sensor recognition systems. *International Journal of Information Fusion (in print*, 2001.

[23] M. M. Kokar, J. A. Tomasik, and J. Weyman. A formal approach to information fusion. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 133–140, 1999.

[24] Z. Korona and M. M. Kokar. Lung sound recognition using model-theory based feature selection. *Applied Signal Processing*, 5:152–169, 1998.

[25] J. Li, M. M. Kokar, and J Weyman. Incorporating uncertainty into the formal development of the fusion operator. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 125–132, 1999.

[26] J. S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice Hall, Inc., 1990.

[27] R. C. Luo and M. G. Kay. Multisensor integration and fusion in intelligent systems. *IEEE Transactions on Systems, Man and Cybernetics*, 19-5:901–931, 1989.

[28] J. A. Makovsky. Why horn formulas matter in computer science: Initial structure and generic examples. In *CAAP'85 Arbres en Algebre et Programmation 10*, pages 374–385, 1985.

[29] J.M. Manyika and H.F. Durrant-White. An information-theoretic approach to management in decentralized data fusion. In *Sensor Fusion V*, volume 1828, pages 202–213. SPIE, 1992.

[30] M. Markin, C. Harris, M. Bernhardt, J. Austin, M. Bedworth, P. Greenway, R. Johnston, A. Little, and D. Lowe. Technology foresight on data fusion and data processing. *Publication of the Royal Aeronautical Society*, 1997.

[31] E. A. Palyutin, J. Saffe, and S. S. Starchenko. Models of superstable horn theories. *Algebra i Logika*, 24:278–326, 1985.

[32] B. C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, 1991.

[33] R. J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[34] N. Saito. *Local Feature Extraction and Its Applications Using a Library of Bases.* PhD thesis, Yale University, 1994.

[35] A. Shulsky. *Silent Warfare: Understanding the World of Intelligence.* Brassey's, 1993.

[36] Y. V. Srinivas. Category theory: Definitions and examples. Technical Report TR-90-14, University of California at Irvine, 1990.

[37] A. N. Steinberg, C. L. Bowman, and F. E. White. Revisions to the jdl data fusion model. In *Proceedings of the SPIE. Sensor Fusion: Architectures, Algorithms and Applications*, pages 430–441. SPIE, 1999.

[38] S. C. A. Thomopoulos. Dignet: a self-organizing neural network for automatic pattern recognition, classification, and data fusion. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 478–495. SPIE, 1992.

[39] J. Tomasik. On products of neat structures. *Annals of Mathematical Logic*, 36:12–16, 1976.

[40] P. K. Varshney. *Distributed Detection and Data Fusion.* Springer-Verlag, 1996.

[41] J. M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 9:8–24, 1990.

[42] L. A. Zadeh. Fuzzy logic. *Computer*, 21, No. 4:83–93, 1988.

Chapter 7

Supplementary Information

7.1 Relevance to the Air Force

The goals of this research are fully within the scope of the AF mission: target detection and recognition. The goal of target detection and recognition is a very difficult task. Currently, image analysts can often detect and identify an enemy aircraft in the imagery, while none of the ATR programs can. This is due to the knowledge that the analysts possess and the ATR programs lack. This research effort's goal is to fill this gap and, ultimately, develop ATR programs that will be able to incorporate background knowledge into their processing, whenever such knowledge becomes available. The same kind of capabilities is extremely important for many civilian applications. These may include facility surveillance systems, drug trafficking counter measures, credit card identification, detection of intruders in computer communication networks, and many others.

7.2 Personnel Supported

The project supported three faculty and four students.

1. Professor Mieczyslaw Kokar (PI)
2. Professor Jerzy Weyman (Co-PI)
3. Professor Jerzy Tomasik (Co-PI)
4. Jingsong Li (M.S. student)
5. Hongge Gao (Ph.D. student)
6. Marek Malczewski (M.S. student)
7. Jiao Wang (Ph.D. student)
8. Gulsah Cakiroglou (M.S. student)

7.3 Publications That Resulted From This Project

[1] M. M. Kokar, M. D. Bedworth, and K. B. Frankel. A reference model for data fusion systems. In Sensor Fusion: Architectures, Algorithms, and Applications IV, Proceedings of the International Conference -AeroSense'2000, vol. 4051, pages 191-202, SPIE, 2000.

[2] J. Tomasik. Discrete dynamic approach to multisensory multitrack fusion. In Proceedings of the International Conference -AeroSense'2000: Sensor Fusion: Architectures, Algorithms, and Applications IV, B. V. Dasarathy (Ed.), vol. 4051, pp. 369-379, 2000.

[3]. M. M. Kokar, J. Weyman and J. A. Tomasik. A Formal Approach to Information Fusion. Information Fusion: An International Journal on Multi-Sensor, Multi-Source Information Fusion, (submitted), 2000.

[4] M. M. Kokar and Z. Korona. A Formal Approach to the Design of Feature-Based Multi-Sensor Recognition Systems. Information Fusion: An International Journal on Multi-Sensor, Multi-Source Information Fusion, (accepted for publication), 2000.

[5] J. Li, M. M. Kokar, J. Weyman and J. Tomasik. Formal Specification of Fuzzy Information Fusion Systems. IEEE Transactions on Systems, Man and Cybernetics, (submitted), 2000.

[6] M. M. Kokar and M. K. Malczewski. Relations among wavelet coefficients and features for ATR. In Sensor Fusion: Architectures, Algorithms, and Applications V, Proceedings of the International Conference -AeroSense'2001, SPIE, 2000.

[7] DeLoach, S. A. and Kokar, M.. Category Theory Approach to Fusion of Wavelet-Based Features. Proceedings of the Second International Conference on Information Fusion, Vol. 1, pages 117-124, 1999.

[8] Kokar, M. M. and Tomasik, J. A. and Weyman, J., A Formal Approach to Information Fusion, Proceedings of the Second International Conference on Information Fusion, Vol. 1, pages 133–140, 1999.

[9] Tomasik, J. A. and Kokar, M. M., Towards a Goal-Driven Autonomous Fusion System, Proceedings of the Second International Conference on Information Fusion, Vol. 1, pages 149–153, 1999.

[10] Gao, H. and Kokar, M. M. and Weyman, J., An Approach to Automation of Fusion Using Specware, Proceedings of the Second International Conference on Information Fusion, Vol. 1, pages 109–116, 1999.

[11] Li, J. and Kokar, M. M. and Weyman, J., Incorporating Uncertainty into the Formal Development of the Fusion Operator, Proceedings of the Second International Conference on Information Fusion, Vol. 1, pages 125–132, 1999.

7.4 Interactions/Transitions

The results of this research have been presented at the SPIE conference AeroSense 2000, Sensor Fusion: Architectures, Algorithms, and Applications IV, Orlando, Florida, April 2000. Another paper will be presented at the SPIE conference in April of 2001. One paper has already been accepted for publication and will appear in International Journal of Information Fusion in 2001.

The PI of this project, Mieczyslaw Kokar, gave two talks on the topics covered by this project at the Air Force Research Laboratory, Rome, NY in 1999 and in 2000.

Air Force Research Laboratory, Rome, Information Systems Design Concepts (IFTD). Contacts: John Lemner and Robert Paragi. Both John Lemner and Robert Paragi have interest in the results of our research. We inform them of our progress.

Chapter 8

Appendix A: Code for Chapter 3

8.1 Rectangle/Triangle Recognition Problem

```
!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% created by Hongge Gao on 04/19/00

spec XX is

 sort Xx

 end-spec

spec YY is

 sort Yy

 end-spec

spec RAN-SENSOR is

   sorts X1, Range
   %% mr - is maximal range
   const mr : Range
   %% f_r - measurement function; returns range for given x
   op f_r: X1 -> Range
   %% O_r? - relation, indicates whether an object is in the sensor's way
   op O_r?: X1, Range -> Boolean
   %% overloading functions
   op lt : Range, Range -> Boolean
   %% object exists only if range is less than max range
   axiom object-exists is
      O_r?(x,f_r(x)) <=> lt(f_r(x),mr)

   %% otherwise, there is no object
   axiom no-object is
     ~O_r?(x,f_r(x)) <=> (f_r(x) = mr)

 end-spec


spec INT-SENSOR is

   sorts X2, Intensity
   %% ishd - value of intensity signal that indicates shadow
   const ishd : Intensity
   %% f_i - measurement function; returns intensity for given x
   op f_i: X2 -> Intensity
   %% S_i? - one-placed relation that indicates where the shadow is
   op S_i?: X2 -> Boolean

   axiom shadow-point is
     (f_i(x) = ishd) => S_i?(x)

 end-spec


spec WORLD is

   %% TRIANGLE,RECTANGLE,TSHADOW are propositions
   %% TRIANGLE - triangle, RECTANGLE - rectangle - objects in our world;
   %% one at a time
   %% TSHADOW - shadow of a triangle

   sorts X, Y, Target

   const TRIANGLE : Target
   const RECTANGLE : Target
   const EMPTY : Target

   op what : X, Y -> Target
   op Tshadow? : X -> Boolean

   %% world theories
   axiom  three-possibilities is
      what(x,y) = TRIANGLE or what(x,y) = RECTANGLE or what(x,y) = EMPTY
   axiom only-one-object is
      ~ ( what(x,y) = TRIANGLE & what(x,y) = RECTANGLE )
   axiom not-tr-shadow-is-rectangle is
      (what(x,y) = TRIANGLE or what(x,y) = RECTANGLE) & ~Tshadow?(x)
        => what(x,y) = RECTANGLE

 end-spec

diagram PROB-DIAGRAM is

 nodes RAN-SENSOR, INT-SENSOR, WORLD, XX, YY
 arcs  XX -> WORLD : { Xx -> X},
       XX -> RAN-SENSOR : { Xx -> X1},
       XX -> INT-SENSOR : { Xx -> X2},
       YY -> RAN-SENSOR : { Yy -> Range},
       YY -> WORLD : { Yy -> Y}

 end-diagram

spec PROB is

  colimit of PROB-DIAGRAM

spec PRE-GOAL is

  import
    translate
       PROB
    by { X1 -> X, Range -> Y}

   op d : (X -> Y), (X -> Intensity) -> ( X, Y -> Target)

 end-spec

diagram GOAL-DIAGRAM is

 nodes TRIV, SET, PRE-GOAL
 arcs  TRIV -> PRE-GOAL : { E -> Target },
       TRIV -> SET : { E -> E }
 end-diagram

spec GOAL is
```

```
  import
    translate
      colimit of GOAL-DIAGRAM
    by { Set -> Set-of-Target,
 empty-set -> empty-set-of-target,
 in -> in-set-of-target,
 E -> Target}

  op dbar : (X -> Y), (X -> Intensity) -> (X,Y -> Set-of-Target)

 end-spec

spec GOAL-WITH-ASSOC is

 import GOAL

 %% association functions

 const a1 : X
 const a2 : X

 %% overloading functions
 op minus : X, X -> X

 op f1p : X -> Y
 definition of f1p is
   axiom f1p(x) = f_r( minus(x,a1))
 end-definition

 op f2p : X -> Intensity
 definition of f2p is
   axiom f2p(x) = f_i( minus(x,a2))
 end-definition

 end-spec
```

8.2 Geometry Ontology

```
!! in-package("SPEC")
!! in-grammar('ispec-grammar)

spec POINT is

 import REAL

 sorts Point

 const origin : Point

 op make-point : Real, Real -> Point

 op x-coord : Point -> Real
 op y-coord : Point -> Real

 axiom x-coord(make-point(x,y)) = x
 axiom y-coord(make-point(x,y)) = y
 axiom make-point(x-coord(p),y-coord(p)) = p

 axiom x-coord(origin) = zero
 axiom y-coord(origin) = zero

 %% projection from a point to x coordinate with angle a is x-y/tan(a)
 op x-p : Point, Real -> Real
 definition of x-p is
   axiom x-p(p,a) = minus(x-coord(p),div(y-coord(p),tan(a)))
 end-definition

 %% distance between two points
 op pt-dis : Point, Point -> Real
 definition of pt-dis is
  axiom pt-dis(p1, p2) = sqrt(plus(square(minus (x-coord(p1),x-coord(p2))),
    square(minus(y-coord(p1),y-coord(p2)))))
 end-definition

 end-spec

spec LINE is

 import POINT
```

```
sort Line
sort CformOfLine

const x-axis : Line
const y-axis : Line

sort-axiom Line = (Real, Real, Real) | line-equation?
sort-axiom CformOfLine = (Real, Real, Real) | canonical-form?

op line-equation? : (Real, Real, Real) -> Boolean
definition of line-equation? is
  axiom line-equation?((a,b,c)) <=> not( a = zero & b = zero)
end-definition

op canonical-form? : (Real, Real, Real) -> Boolean
definition of canonical-form? is
  axiom canonical-form?((a,b,c)) <=>
   ( b = one) or ( a = one & b = zero)
end-definition

%% a straight line is of the form ax + by + c = 0
op make-line : Real, Real, Real -> Line

%% the canonical form of a line
%% if b = 0, the canonical form is : x + c/a = 0 (1,0, c/a)
%% if b != 0, the canonical form is : ax/b+y+c/b=0 (a/b,1,c/b)
op cl-form : Line ->  CformOfLine
definition of cl-form is
  axiom ~( b-par(ln) = zero ) =>
      relax(canonical-form?)(cl-form(ln)) =
        (div(a-par(ln),b-par(ln)), one, div(c-par(ln),b-par(ln)))
  axiom ( b-par(ln) = zero & ~( a-par(ln) = zero )) =>
      relax(canonical-form?)(cl-form(ln)) =
        (one, zero, div(c-par(ln), a-par(ln)))
end-definition

op line-form : CformOfLine, Nonzero -> Line
definition of line-form is
 axiom line-form(cl,r) =
       make-line(times(relax(nonzero?)(r),project(1)(relax(canonical-form?)(cl))),
 times(relax(nonzero?)(r),project(2)(relax(canonical-form?)(cl))),
 times(relax(nonzero?)(r),project(3)(relax(canonical-form?)(cl))))
end-definition

%% equivalence relation between lines
```

```
op same-line? : Line, Line -> Boolean
definition of same-line? is
  axiom same-line?(ln1,ln2) <=> cl-form(ln1) = cl-form(ln2)
end-definition

%% from line to a, b, c parameters
op a-par : Line -> Real
definition of a-par is
  axiom a-par(ln) = project(1)(relax(line-equation?)(ln))
end-definition

op b-par : Line -> Real
definition of b-par is
  axiom b-par(ln) = project(2)(relax(line-equation?)(ln))
end-definition

op c-par : Line -> Real
definition of c-par is
  axiom c-par(ln) = project(3)(relax(line-equation?)(ln))
end-definition

axiom a-par(x-axis) = zero
axiom c-par(x-axis) = zero
axiom b-par(y-axis) = zero
axiom c-par(y-axis) = zero

%% a point on the line if it satisfies the line equation, i.e.,
%% ax + by + c = 0 where
%% x, y are x and y coordinates of the point and
%% a, b, c are the three parameters that make the line
op pt-on-line? : Point, Line -> Boolean
definition of pt-on-line? is
  axiom pt-on-line?(make-point(x,y), make-line(a,b,c)) <=>
      plus(plus(times(a,x),times(b,y)),c) = zero
end-definition

%% two points p1=(x1,y1), p2=(x2,y2) are on the same side of line = (a,b,c)
%% if ax1 + by1 + c > 0 and ax2 + by2 + c > 0
%% or ax1 + by1 + c < 0 and ax2 + by2 + c < 0
op same-side-of-line? : Point, Point, Line -> Boolean
definition of same-side-of-line? is
  axiom same-side-of-line?(make-point(x1,y1),make-point(x2,y2),make-line(a,b,c)) <=>
      ( gt(plus(plus(times(a,x1),times(b,y1)),c),zero) &
gt(plus(plus(times(a,x2),times(b,y2)),c),zero) ) or
      ( lt(plus(plus(times(a,x1),times(b,y1)),c),zero) &
lt(plus(plus(times(a,x2),times(b,y2)),c),zero) )
end-definition

%% slope of straight line is -a/b
op slope : Line -> Real
definition of slope is
  axiom
    ln = make-line(a,b,c) & b = zero => slope(ln) = infinity
  axiom
    ln = make-line(a,b,c) & ~(b = zero) => slope(ln) = negate(div(a,b))
end-definition

%% intersection with x-axis (or x-intercept) is x=-c/a
op x-cept : Line -> Real
definition of x-cept is
  axiom x-cept(make-line(a,b,c)) = negate(div(c,a))
end-definition

%% intersection with y-axis (or y-intercept) is y= -c/b
op y-cept : Line -> Real
definition of y-cept is
  axiom y-cept(make-line(a,b,c)) = negate(div(c,b))
end-definition

%% line is parallel to the x-axis if a=0
op x-pl? : Line -> Boolean
definition of x-pl? is
  axiom x-pl?(p) <=> a-par(p) = zero
end-definition

%% line is paralle to the y-axis if b=0
op y-pl? : Line -> Boolean
definition of y-pl? is
  axiom y-pl?(p) <=> b-par(p) = zero
end-definition

theorem x-pl?(x-axis)
theorem y-pl?(y-axis)


%% two lines are parallel when a1xb2=a2xb1
op pl? : Line, Line -> Boolean
definition of pl? is
axiom pl?(ln1,ln2) <=> times(a-par(ln1),b-par(ln2)) = times(a-par(ln2),b-par(ln1))
end-definition
```

```
 %% two lines are perpendicular when a1xa2 = -b1xb2
 op pp? : Line, Line -> Boolean
 definition of pp? is
 axiom pp?(ln1,ln2) <=>
     times(a-par(ln1),a-par(ln2)) = negate(times(b-par(ln1),b-par(ln2)))
 end-definition

 %% the angle between two lines
 op angle: Line, Line -> Real
 definition of angle is
   axiom pp?(ln1,ln2) => angle(ln1,ln2) = nat-to-real(90)
   axiom ~pp?(ln1,ln2) =>
      angle(ln1,ln2) = arctan(div(minus(times(a-par(ln1),b-par(ln2)),
times(a-par(ln2),b-par(ln1))),
  plus(times(a-par(ln1),a-par(ln2)),
       times(b-par(ln1),b-par(ln2)))))
 end-definition

 theorem fa(ln1:Line, ln2:Line) pl?(ln1,ln2) <=> angle(ln1,ln2) = zero
 theorem fa(ln1:Line, ln2:Line) pp?(ln1,ln2) <=> angle(ln1,ln2) = nat-to-real(90)

 theorem fa(ln1:Line, ln2:Line) pl?(ln1,ln2) <=> slope(ln1) = slope(ln2)
 theorem fa(ln1:Line, ln2:Line) pp?(ln1,ln2) <=> times(slope(ln1),slope(ln2)) = negat

 end-spec

spec SEGMENT is

 import LINE

 sorts Segment

 op make-segment : Point, Point -> Segment
 op start-point : Segment -> Point
 op end-point : Segment -> Point

 axiom start-point(make-segment(s,e)) = s
 axiom end-point(make-segment(s,e)) = e
 axiom make-segment(start-point(s),end-point(s)) = s

 %% from segment to line
 %% if start point and end point have same x coordinates,
 %% i.e., start-point = (x1,y1), end-point = (x2,y2) and x1=x2
 %% then the canonical form of the line is x - x1 = 0(one, zero, -x1)
%% otherwise, the canonical form of the line is
%% y = ((y1-y2)/(x1-x2))x + (y1-(y1-y2)x1/(x1-x2)) (a-part,-1,c-part)
op seg-to-line : Segment -> Line
definition of seg-to-line is
  axiom x-coord(start-point(sg)) = x-coord(end-point(sg)) =>
    seg-to-line(sg) = make-line(one,zero,negate(x-coord(start-point(sg))))
  axiom ~(x-coord(start-point(sg)) = x-coord(end-point(sg))) =>
    seg-to-line(sg) = make-line(a-part(sg),b-part(sg),c-part(sg))
end-definition

op a-part : Segment -> Real
definition of a-part is
  axiom a-part(sg) = div(minus(y-coord(start-point(sg)),y-coord(end-point(sg))),
 minus(x-coord(start-point(sg)),x-coord(end-point(sg))))
end-definition

op b-part : Segment -> Real
definition of b-part is
  axiom b-part(sg) = negate(one)
end-definition

op c-part : Segment -> Real
definition of c-part is
 axiom c-part(sg) = minus(y-coord(start-point(sg)),
  times(a-part(sg),x-coord(start-point(sg))))
end-definition

%% permutation of two pairs of points
op permutation? : (Point, Point), (Point, Point) -> Boolean
definition of permutation? is
  axiom permutation?((p1,p2),(p3,p4)) <=>
     ( p1 = p3 & p2 = p4 ) or ( p1 = p4 & p2 = p3 )
end-definition

op congruent-segment? : Segment, Segment -> Boolean
definition of congruent-segment? is
  axiom congruent-segment?(make-segment(s1,e1),make-segment(s2,e2)) <=>
     permutation?((s1,e1),(s2,e2))
end-definition

%% slope of segment is defined in terms of slope of line
op slope : Segment -> Real
definition of (slope: (Segment -> Real)) is
 axiom slope(sg) = (slope : (Line -> Real))(seg-to-line(sg))
end-definition
```

```
 %% angle of segments is also defined in terms of slope of lines
 op angle : Segment, Segment -> Real
 definition of (angle: (Segment, Segment -> Real)) is
   axiom angle(s1,s2) = (angle: (Line, Line -> Real))(seg-to-line(s1),seg-to-line(s2)
 end-definition

 %% predicates of parallel or perpendicular
 op pl? : Segment, Segment -> Boolean
 definition of (pl?: (Segment, Segment -> Boolean)) is
   axiom pl?(s1,s2) <=> (pl?: (Line, Line -> Boolean))(seg-to-line(s1),seg-to-line(s2
 end-definition

 op pp? : Segment, Segment -> Boolean
 definition of (pp?: (Segment, Segment -> Boolean)) is
   axiom pp?(s1,s2) <=> (pp?: (Line, Line -> Boolean))(seg-to-line(s1),seg-to-line(s2
 end-definition

 %% length of a segment
 op length : Segment -> Real
 definition of length is
  axiom length(make-segment(s,e)) = pt-dis(s,e)
 end-definition

 %% point on segment
 op pt-on-seg? : Point, Segment -> Boolean
 definition of pt-on-seg? is
   axiom pt-on-seg?(p,s) <=>
       pt-on-line?(p,seg-to-line(s)) &
       leq(pt-dis(p,start-point(s)),length(s)) &
       leq(pt-dis(p,end-point(s)),length(s))
 end-definition

 end-spec

spec TRIANGLE is

 import SEGMENT

 sort Triangle

 op make-triangle : Point, Point, Point -> Triangle
 op pA : Triangle -> Point
 op pB : Triangle -> Point
 op pC : Triangle -> Point
```

```
axiom pA(make-triangle(A,B,C)) = A
axiom pB(make-triangle(A,B,C)) = B
axiom pC(make-triangle(A,B,C)) = C

%% three sides of triangle
op side-a : Triangle -> Segment
definition of side-a is
 axiom side-a(make-triangle(A,B,C)) = make-segment(B,C)
end-definition

op side-b : Triangle -> Segment
definition of side-b is
 axiom side-b(make-triangle(A,B,C)) = make-segment(C,A)
end-definition

op side-c : Triangle -> Segment
definition of side-c is
 axiom side-c(make-triangle(A,B,C)) = make-segment(A,B)
end-definition

%% three angles of a triangle
op angle-a : Triangle -> Real
definition of angle-a is
  axiom angle-a(tri) = angle(side-b(tri),side-c(tri))
end-definition

op angle-b : Triangle -> Real
definition of angle-b is
  axiom angle-b(tri) = angle(side-a(tri),side-c(tri))
end-definition

op angle-c : Triangle -> Real
definition of angle-c is
  axiom angle-c(tri) = angle(side-a(tri),side-b(tri))
end-definition

%% an angle is acute when it is less than ninety degrees
op acute? : Real -> Boolean
definition of acute? is
 axiom acute?(a) <=> lt(a, nat-to-real(90))
end-definition

%% an angle is right when it is equal to 90 degrees
op right? : Real -> Boolean
```

```
definition of right? is
 axiom right?(a) <=> a = nat-to-real(90)
end-definition

%% an angle is obtuse when it is larger than 90 degrees
op obtuse? : Real -> Boolean
definition of obtuse? is
  axiom obtuse?(a) <=> gt(a,nat-to-real(90))
end-definition

%% acute triangle: all angles are acute
op acute-triangle? : Triangle -> Boolean
definition of acute-triangle? is
  axiom acute-triangle?(tri) <=>
     acute?(angle-a(tri)) &
     acute?(angle-b(tri)) &
     acute?(angle-c(tri))
end-definition

%% right triangle: has one right angle
op right-triangle? : Triangle -> Boolean
definition of right-triangle? is
  axiom right-triangle?(tri) <=>
    right?(angle-a(tri)) or
    right?(angle-b(tri)) or
    right?(angle-c(tri))
end-definition

%% obtuse-triangle : has one obtuse angle
op obtuse-triangle? : Triangle -> Boolean
definition of obtuse-triangle? is
  axiom obtuse-triangle?(tri) <=>
    obtuse?(angle-a(tri)) or
    obtuse?(angle-b(tri)) or
    obtuse?(angle-c(tri))
end-definition

%% axioms of triangle
axiom plus(plus(angle-a(tri),angle-b(tri)),angle-c(tri)) = nat-to-real(180)
axiom law-of-sines-1 is
   div(length(side-a(tri)),sin(angle-a(tri))) =
   div(length(side-b(tri)),sin(angle-b(tri)))
axiom law-of-sines-2 is
   div(length(side-a(tri)),sin(angle-a(tri))) =
   div(length(side-c(tri)),sin(angle-c(tri)))
```

```
%% permutaton of a pair of triple points
op permutation? : (Point,Point,Point),(Point,Point,Point) -> Boolean
definition of
 (permutation? : ((Point,Point,Point),(Point,Point,Point) -> Boolean)) is
  axiom permutation?((s1,s2,s3),(e1,e2,e3)) <=>
    (s1 = e1 & s2 = e2 & s3 = e3) or
    (s1 = e1 & s2 = e3 & s3 = e2) or
    (s1 = e2 & s2 = e1 & s3 = e3) or
    (s1 = e2 & s2 = e3 & s3 = e1) or
    (s1 = e3 & s2 = e1 & s3 = e2) or
    (s1 = e3 & s2 = e2 & s3 = e1)
end-definition

%% two triangles are congruent if vertices are permutation of each other.
op congruent-triangle? : Triangle, Triangle -> Boolean
definition of congruent-triangle? is
 axiom congruent-triangle?(make-triangle(s1,s2,s3),make-triangle(e1,e2,e3)) <=>
     permutation?((s1,s2,s3),(e1,e2,e3))
end-definition

op pt-inside-triangle? : Point, Triangle -> Boolean
definition of pt-inside-triangle? is
  axiom pt-inside-triangle?(p,t) <=>
      same-side-of-line?(p,pA(t),seg-to-line(side-a(t))) &
      same-side-of-line?(p,pB(t),seg-to-line(side-b(t))) &
      same-side-of-line?(p,pC(t),seg-to-line(side-c(t)))
end-definition

end-spec


spec QUADRILATERALS is

 import  SEGMENT

 sort Quad

 sort-axiom Quad = (Point, Point, Point, Point) | quad?

 op quad? : (Point, Point, Point, Point) -> Boolean
 definition of quad? is
   axiom quad?((A, B, C, D)) <=>
     not ( ex (p : Point) (pt-on-seg?(p,make-segment(A,B)) &
   pt-on-seg?(p,make-segment(C,D)))) &
```

```
    not(pt-on-line?(C,seg-to-line(make-segment(A,B)))) &
    not(pt-on-line?(D,seg-to-line(make-segment(B,C)))) &
    not(pt-on-line?(A,seg-to-line(make-segment(C,D)))) &
    not(pt-on-line?(B,seg-to-line(make-segment(D,A))))
end-definition

op make-quad : Point, Point, Point, Point -> Quad
op pA : Quad -> Point
op pB : Quad -> Point
op pC : Quad -> Point
op pD : Quad -> Point

axiom pA(make-quad(A,B,C,D)) = A
axiom pB(make-quad(A,B,C,D)) = B
axiom pC(make-quad(A,B,C,D)) = C
axiom pD(make-quad(A,B,C,D)) = D

axiom make-quad(pA(q),pB(q),pC(q),pD(q)) = q

%% four sides of quadrilaterals
op side-a : Quad -> Segment
definition of (side-a:(Quad -> Segment)) is
 axiom side-a(q) = make-segment(pA(q),pB(q))
end-definition

op side-b : Quad -> Segment
definition of (side-b:(Quad -> Segment)) is
 axiom side-b(q) = make-segment(pB(q),pC(q))
end-definition

op side-c : Quad -> Segment
definition of (side-c:(Quad -> Segment)) is
 axiom side-c(q) = make-segment(pC(q),pD(q))
end-definition

op side-d : Quad -> Segment
definition of (side-d:(Quad -> Segment)) is
 axiom side-d(q) = make-segment(pD(q),pA(q))
end-definition

%% four angles of quadrilaterals
op angle-a : Quad -> Real
definition of (angle-a:(Quad -> Real)) is
  axiom angle-a(q) = angle(side-a(q),side-d(q))
end-definition
```

```
 op angle-b : Quad -> Real
 definition of (angle-b:(Quad -> Real)) is
   axiom angle-b(q) = angle(side-a(q),side-b(q))
 end-definition

 op angle-c : Quad -> Real
 definition of (angle-c:(Quad -> Real)) is
   axiom angle-c(q) = angle(side-b(q),side-c(q))
 end-definition

 op angle-d : Quad -> Real
 definition of (angle-d:(Quad -> Real)) is
   axiom angle-d(q) = angle(side-c(q),side-d(q))
 end-definition

 axiom plus(plus(angle-a(q),angle-b(q)),
    plus(angle-c(q),angle-d(q))) = nat-to-real(360)

 end-spec

spec PARALLELOGRAM is

 import QUADRILATERALS

 sort PGram

 sort-axiom PGram = Quad | pgram?

 %% in a parallelogram, opposite sides are parallel
 op pgram? : Quad -> Boolean
 definition of pgram? is
  axiom pgram?(q) <=>
      pl?(side-a(q),side-c(q)) & pl?(side-b(q),side-d(q))
 end-definition

 %% in a parallelogram, opposite sides have the same length
 axiom fa(p:PGram)
    (length(side-a(relax(pgram?)(p))) = length(side-c(relax(pgram?)(p))) &
     length(side-b(relax(pgram?)(p))) = length(side-d(relax(pgram?)(p))))

 %% in a parallelogram, two consecutive angles add up to 180 degrees.
 axiom (plus(angle-a(relax(pgram?)(p)),angle-b(relax(pgram?)(p)))
= nat-to-real(180))
 axiom (plus(angle-b(relax(pgram?)(p)),angle-c(relax(pgram?)(p)))
= nat-to-real(180))
 axiom (plus(angle-c(relax(pgram?)(p)),angle-d(relax(pgram?)(p)))
= nat-to-real(180))
 axiom (plus(angle-d(relax(pgram?)(p)),angle-a(relax(pgram?)(p)))
= nat-to-real(180))

 end-spec

spec RECTANGLE is

 import PARALLELOGRAM

 sort Rectangle

 sort-axiom Rectangle = PGram | rectangle?

 %% rectangle is a parallelogram in which all angles equal 90 degrees
 op rectangle? : PGram -> Boolean
 definition of rectangle? is
  axiom rectangle?(p) <=>
     angle-a(relax(pgram?)(p)) = nat-to-real(90) &
     angle-b(relax(pgram?)(p)) = nat-to-real(90) &
     angle-c(relax(pgram?)(p)) = nat-to-real(90) &
     angle-d(relax(pgram?)(p)) = nat-to-real(90)
 end-definition

 end-spec

spec RHOMBUS is

 import PARALLELOGRAM

 sort Rhombus

 sort-axiom Rhombus = PGram | rhombus?

 %% a rhombus is a parallelogram in which adjacent sides have the same length
 op rhombus? : PGram -> Boolean
 definition of rhombus? is
  axiom rhombus?(p) <=>
      side-a(relax(pgram?)(p)) = side-b(relax(pgram?)(p))
 end-definition

 theorem fa(r:Rhombus)
    (side-b(relax(pgram?)(relax(rhombus?)(r))) =
    side-c(relax(pgram?)(relax(rhombus?)(r))))
theorem fa(r:Rhombus)
   (side-c(relax(pgram?)(relax(rhombus?)(r))) =
    side-d(relax(pgram?)(relax(rhombus?)(r))))
theorem fa(r:Rhombus)
   (side-d(relax(pgram?)(relax(rhombus?)(r))) =
    side-a(relax(pgram?)(relax(rhombus?)(r))))

end-spec
```

8.3 Ground-Based Low PRF Radar

```
!! in-package("SPEC")
!! in-grammar('ISPEC-GRAMMAR)

%% Specification for Ground-Based Low PRF Radar

spec GB_LPRF_RADAR is

 import REAL

 sorts Radar, Target, Range, Angle

 sort-axiom Range = Real
 sort-axiom Angle = Real

 % light speed constant
 const c : Real
 const two : Real
 const four : Real
 const sixteen : Real
 const pi : Real
 % Boltzman's constant
 const K : Real
 % transmit and received losses
 const L : Real
 % noise temperature
 const T : Real
 %% overloading operations
 op div : Real, Real -> Real

 % basic arithematic operations
 op sq : Real -> Real
 definition of sq is
   axiom sq(a) = times(a, a)
 end-definition
 op cube : Real -> Real
 definition of cube is
  axiom cube(a) = times(a, sq(a))
 end-definition
 op sqr : Real -> Real

 %% basic radar parameters
 % radar aperture
```

```
op ap : Radar -> Real
% radar wavelength
op wl : Radar -> Real
% interpulse period
op ipp : Radar -> Real
% radar dimension
op dim : Radar -> Real
% radar pulse length
op rpl : Radar -> Real
% signal-noise ratio threshold for detection
op sn_thres : Radar -> Real
% Power radiated by the radar
op pow : Radar -> Real
% Receiver noise bandwidth
op bn : Radar -> Real
% noise figure
op nf : Radar -> Real
% scan space
op ss : Radar -> Real
% scan time
op st : Radar -> Real

%% radar derived parameters
% beamwidth of radar
op bw : Radar -> Angle
definition of bw is
  axiom bw(ra) = div(wl(ra), dim(ra))
end-definition
% radar gain
op gain : Radar -> Real
definition of gain is
 axiom gain(ra) = div(times(four,pi), sq(bw(ra)))
end-definition
% radar frequency
op freq : Radar -> Real
definition of freq is
 axiom freq(ra) = div(c,wl(ra))
end-definition
% average radiated power
op apow : Radar -> Real
definition of apow is
 axiom apow(ra) = div(times(pow(ra),rpl(ra)),ipp(ra))
end-definition

%% -- Radar target attributes --
%% Radar target cross section
op rcs : Target -> Real
%% target azimuth and elevation coordinate
op azimuth : Target -> Real
op elevation : Target -> Real
%% elapsed time between transmitted and received signal
op et : Target -> Real


%% range function
op range : Target -> Range
definition of range is
  axiom range(ta) = div(times(c, et(ta)), two)
end-definition

%% maximum range a radar can detect
op max_range : Radar -> Range
definition of max_range is
  axiom max_range(ra) = div(times(c, ipp(ra)), two)
end-definition

%% maximum range a radar can detect for a specific target
op mtr : Radar, Target -> Range
definition of mtr is
 axiom mtr(ra,tar) =
  sqr(sqr(div(times(times(apow(ra),ap(ra)),
                  times(rcs(tar),st(ra))),
             times(times(times(sixteen,K),T),
                  times(times(L,sn_thres(ra)),ss(ra))))))
end-definition

%% target power received at the radar.
op pr : Radar, Target, Range -> Real
definition of pr is
 axiom pr(ra,tar,r) =
   div(times(times(pow(ra),sq(gain(ra))),
            times(sq(wl(ra)),rcs(tar))),
      times(cube(times(four,pi)),times(sq(r),sq(r))))
end-definition

%% total receiver noise power
op noise : Radar -> Real
definition of noise is
 axiom noise(ra) = times(times(nf(ra),K),times(T,bn(ra)))
end-definition
```

```
op sn_ratio : Radar, Target, Range -> Real
definition of sn_ratio is
 axiom sn_ratio(ra,tar,r) =
    div(pr(ra,tar,r),times(noise(ra),L))
end-definition

%% range resolution
op rr : Radar -> Range
definition of rr is
  axiom rr(ra) = div(times(c, rpl(ra)), four)
end-definition

%% angluar resolution
op ar : Radar -> Angle
definition of ar is
  axiom ar(ra) = div(bw(ra), two)
end-definition

%% radar range deviation
op rrd : Radar, Target, Range -> Range
definition of rrd is
 axiom rrd(ra,tar,r) = div(rr(ra),sqr(sn_ratio(ra,tar,r)))
end-definition

%% radar angular deviation
op rad : Radar, Target, Range -> Angle
definition of rad is
 axiom rad(ra,tar,r) = div(ar(ra), sqr(sn_ratio(ra,tar,r)))
end-definition

%% radar uncertainty measurement
op uncertain : Radar, Target, Range -> Real
definition of uncertain is
 axiom uncertain(ra,tar,r) = div(one,times(rrd(ra,tar,r),two))
end-definition

%% axioms
% maximum range axiom
axiom max_range is
  fa (target, radar) leq( mtr(radar,target), max_range(radar))
% maximum target range axiom
axiom max_target_range is
  fa (target, radar) leq( range(target), mtr(radar,target))
end-spec
```

8.4 FLIR

```
!! in-package("SPEC")
!! in-grammar('ISPEC-GRAMMAR)

spec FLIR is

  import REAL

  sorts Flir, Target, Range, Angle
  sort-axiom Angle = Real
  sort-axiom Range = Real

  const Four : Real
  const Pi   : Real


  %% basic overloading operations
  op div : Real, Real -> Real
  op sq : Real -> Real
  definition of sq is
    axiom sq(a) = times(a,a)
  end-definition
  op sqr : Real -> Real

  %% basic FLIR constants
  const Kfd  : Real   %% bandwidth proportionality constant
  const Ko   : Real   %% optics loss
  const Kc   : Real   %% electronics loss
  const Kd   : Real   %% detector loss
  const Km   : Real   %% monitor loss

  %% basic FLIR parameters
  % diameter of optical aperture
  op dia : Flir -> Real
  % instantaneous field of view of each detector
  op IFOV : Flir -> Real
  % focal number
  op fn : Flir -> Real
  % solid angle coverage of the sensor
  op omega : Flir -> Real
  % frame time
  op Tf : Flir -> Real
  % number of detectors in parallel
```

```
op nod : Flir -> Real
% total scanning loss factor
op Kn : Flir -> Real
% detector detectivity
op dstar : Flir -> Real

%% basic target parameters
% radiation intensity
op Jt : Target -> Real
% target range
op range : Target -> Range

%% Performance (for point source target radiating in a narrow bandwidth)
% irradiance at the entrance aperture of the optics
op Ht : Target -> Real
definition of ht is
  axiom Ht(tar) = div(jt(tar), sq(range(tar)))
end-definition
% signal power at the optical aperture
op spow : Flir, Target -> Real
definition of spow is
 axiom spow(fl,tar) = times(div(times(Pi,sq(dia(fl))), Four), Ht(tar))
end-definition
% detector area
op Ad : Flir -> Real
definition of Ad is
  axiom Ad(fl) = times(IFOV(fl),sq(times(fn(fl),dia(fl))))
end-definition
% number of resolution elements
op Nr : Flir -> Real
definition of Nr is
  axiom Nr(fl) = div(omega(fl), IFOV(fl))
end-definition
% on-target time per resolution cell
op Td : Flir -> Real
definition of Td is
  axiom Td(fl) = times(times(nod(fl), Kn(fl)),div(Tf(fl), Nr(fl)))
end-definition
% equivalent filter noise bandwidth
op df : Flir -> Real
definition of df is
  axiom df(fl) = div(Kfd, Td(fl))
end-definition
% detector noise level
op npow : Flir -> Real
```

```
 definition of npow is
   axiom npow(fl) = div(sqr(times(Ad(fl), df(fl))), dstar(fl))
 end-definition
 % system modulation transfer function (MTF)
 op MTF : Flir -> Real
 definition of MTF is
   axiom MTF(fl) = times(times(ko,Kc),times(Kd,Km))
 end-definition
 % system loss factor
 op loss : Flir -> Real
 definition of loss is
   axiom loss(fl) = times(MTF(fl), sqr(div(Kn(fl),Kfd)))
 end-definition
 % signal-noise ratio
 op sn_ratio : Flir, Target -> Real
 definition of sn_ratio is
   axiom sn_ratio(fl, tar) = times(div(spow(fl,tar),npow(fl)), loss(fl))
 end-definition


end-spec
```

8.5 Ladar

```
!! in-package("SPEC")
!! in-grammar('ISPEC-GRAMMAR)

%% Laser radar systems are active devices that operate in a manner very
%% similar to microwave radars but at a much higher frequency. This higher
%% frequency allows using smaller components, and has remarkable angular
%% accuracy. The atmospheric attenuation losses, however, are considerable
%% at these high frequencies.

spec LADAR is

  import REAL

  sorts Ladar, Target, Range, Angle
  sort-axiom Range = Real
  sort-axiom Angle = Real

  const two  : Real
  const Pi   : Real
  const four : Real
  const C    : Real   %% light speed
  const Pc   : Real   %% Plank's constant

  %% basic new and overloading operations of Real
  op times3 : Real, Real, Real -> Real
  definition of times3 is
    axiom times3(a,b,c) = times(times(a,b), c)
  end-definition

  op times4 : Real, Real, Real, Real -> Real
  definition of times4 is
    axiom times4(a,b,c,d) = times(times(a,b), times(c,d))
  end-definition

  op div : Real, Real -> Real
  op sq  : Real -> Real
  definition of sq is
    axiom sq(a) = times(a,a)
  end-definition

  %% basic target parameters
```

```
% target backscattering coefficient
op tbc : Target -> Real
% distribution of incident power assumption
op Lambertian : Target -> Boolean
% radar cross section (RCS)
op rcs : Target -> Real
% small or large target
op small : Target, Ladar -> Boolean
% elapsed time between transmit and receive pulses
op et : Target -> Real
% target angular position
op az : Target -> Angle
op el : Target -> Angle

%% basic laser radar parameters
% transmitted pulsed power
op pow : Ladar -> Real
% lens area
op area : Ladar -> Real
% optical efficiency
op oe : Ladar -> Real
% ladar beamwidth
op bw : Ladar -> Angle
% ladar wavelength
op rwl : Ladar -> Real
% ladar dimension
op dim : Ladar -> Real
% Ladar frequency
op freq . Ladar -> Real
% receiver bandwidth
op rb : Ladar -> Real
% quantum efficiency
op qe : Ladar -> Real

axiom beamwidth_wavelength_dimension is
  bw(la) = div(rwl(la), dim(la))
axiom radar_area_and_dimension is
  area(la) = div(times(Pi, sq(dim(la))), four)
axiom radar_wavelength_and_frequency is
  rwl(la) = div(C, freq(la))

%% measurement calculations
% distance between radar and target
op range : Target -> Range
definition of range is
```

```
    axiom range(tar) = div(times(C, et(tar)), two)
  end-definition


  %% Performance calculations
  % signal-noise ratio
  op spow1 : Ladar, Target -> Real
  definition of spow1 is
   axiom spow1(la,tar) = div(times4(pow(la),area(la),oe(la),tbc(tar)),
     times(Pi, sq(range(tar))))
  end-definition
  op spow2 : Ladar, Target -> Real
  definition of spow2 is
    axiom spow2(la,tar) = div(times3(spow1(la,tar),two,rcs(tar)),
      times3(Pi,sq(range(tar)), sq(bw(la))))
  end-definition
  op spow : Ladar, Target -> Real
  definition of spow is
    axiom Lambertian(tar) => spow(la, tar) = spow1(la, tar)
    axiom ~Lambertian(tar) => spow(la, tar) = div(spow1(la,tar), two)
    axiom small(tar,la) => spow(la,tar) = spow2(la, tar)
  end-definition
  op noise : Ladar -> Real
  definition of noise is
    axiom noise(la) = times(div(times(freq(la), Pc), qe(la)), rb(la))
  end-definition
  op sn_ratio : Ladar, Target -> Real
  definition of sn_ratio is
    axiom sn_ratio(la,tar) = div(spow(la,tar), noise(la))
  end-definition

end-spec
```

8.6 MMW Radar

```
!! in-package("SPEC")
!! in-grammar('ISPEC-GRAMMAR)

%% MMW radar systems for ground applications are constructed at frequencies
%% of 35, 95, 140, and 220 GHz.
%% MMW radar systems are used to detect nonmoving targets, which can be
%% stationary ground targets, such as tanks and trucks, or airborne hovering
%% helicopters, and the detection could take place day or night, in the
%% presence of ground clutter return, and under adverse weather conditions.

spec MMW_RADAR is

 import REAL

 sorts Radar, Target, Range, Angle
 sort-axiom Range = Real
 sort-axiom Angle = Real

 %% constants
 const C : Real   %% light speed
 const two : Real

 %% overloading operations
 op div : Real, Real -> Real
 op leq : Real, Real -> Boolean

 %% basic MMW radar parameters
 % radar wavelength
 op rwl : Radar -> Real
 % peak power
 op ppow : Radar -> Real
 % radar pulse width
 op rpw : Radar -> Real
 % pulse repeat frequency
 op prf : Radar -> Real
 % radar antenna diameter
 op rad : Radar -> Real
 % radar antenna aperture
 op raa : Radar -> Real
 % radar frametime
 op rft : Radar -> Real
 % radar equivalent noise temperature
```

```
op rent : Radar -> Real
% losses
op loss : Radar -> Real
% azimuth scan
op ras : Radar -> Angle
% elevation scan
op res : Radar -> Angle
% target-to-clutter signal ratio for detection
op tc_ratio : Radar -> Real
% backscattering cross section
op bcs : Radar -> Real

%% derived radar parameters
% average power
op apow : Radar -> Real
definition of apow is
 axiom apow(ra) = times(times(ppow(ra), rpw(ra)), prf(ra))
end-definition
% radar beamwidth
op bw : Radar -> Angle
definition of bw is
  axiom bw(ra) = div(rwl(ra), rad(ra))
end-definition

%% -- Radar target attributes --
%% Radar target cross section
op rcs : Target -> Real
%% target azimuth and elevation coordinate
op azimuth : Target -> Real
op elevation : Target -> Real
%% elapsed time between transmitted and received signal
op et : Target -> Real

%% radar measurements
op range : Target -> Range
definition of range is
  axiom range(ta) = div(times(C, et(ta)), two)
end-definition

%% performance of radar
% range resulution
op rr : Radar -> Range
definition of rr is
  axiom rr(ra) = div(times(C, rpw(ra)), two)
end-definition
```

```
% detection range of nonmoving target
op rdr : Radar, Target -> Range
definition of rdr is
  axiom rdr(ra,t) = div(rcs(t), times(times(rr(ra),bw(ra)),
                                      times(tc_ratio(ra),bcs(ra))))
end-definition

%% detection range axiom
axiom detection_range is
  fa(tar) leq(range(tar), rdr(ra, tar))

end-spec
```

8.7 SAR

```
!! in-package("SPEC")
!! in-grammar('ISPEC-GRAMMAR)

spec SAR is

 import Real

 sorts Sar, Target, Range, Angle

 sort-axiom Range = Real
 sort-axiom Angle = Real
 const C : Real
 const two : Real

 %% overloading operations
 op div : Real, Real -> Real
 op cos : Real -> Real
 op sin : Real -> Real

 %% basic SAR radar parameters
 % airplane velocity
 const V : Real
 % doppler shifts resolution
 op doppler : Sar -> Real
 % radar pulse length
 op rpl : Sar -> Real
 % depression angle
 op alpha : Sar -> Angle
 % subtended angle between the velocity vector and R
 op theta : Sar -> Angle
 % pluse compression ratio
 op pc : Sar -> Real
 % sar radar wavelength
 op wl : Sar -> Real

 %% basic target attributes
 % elapsed time
 op et : Target -> Real
 % x, y coordinates
 op x-coor : Target -> Real
 op y-coor : Target -> Real
```

```
%% range function
op range : Target -> Range
definition of range is
 axiom range(tar) = div(times(C, et(tar)), two)
end-definition

%% resulution functions
% range resolution
op rr : Sar -> Range
definition of rr is
  axiom rr(sar) = div(times(C, rpl(sar)), two)
end-definition
% x direction resolution
op xr : Sar -> Range
definition of xr is
  axiom xr(sar) = div(div(rr(sar), cos(alpha(sar))), pc(sar))
end-definition
% y direction resolution
op yr : Sar, Range -> Range
definition of yr is
 axiom yr(sar,r) = div(times(times(r,wl(sar)), doppler(sar)),
                       times(times(two, V), sin(theta(sar))))
end-definition

end-spec
```

Chapter 9

Appendix B: Code for Chapter 4

The fuzzy specification library contains the following specs :

```
fuzzy set:  fs.re
fuzzy number: fs.re
fuzzy arithmetic: fa.re
fuzzy reasoning: fr.re
image: image.re
fuzzification: ff.re
tri-fuzzify: triff.re
tra-fuzzify: traff.re
gua-fuzzify: guaff.re
defuzzification: defuzzy.re
defuzzification-2: defuzzy2.re
edge point : epoint.re
edge : edge.re
fuzzy edge point: fepoint.re
```

```
%% Formal Information Fusion Library Specs
%% Fuzzy edge point specification: FUZZY-EDGE
%% Author: Jingsong Li
%% Created: Aug. 18, 1998
%% Last updated: Nov. 2, 1999
%% Comments: For a long time I have been stuck at how to define
%% fuzzy logic operations such as fuzzy equal, fuzzy less than
%% and fuzzy great than.
%% Modified: July 8, 2000
%% Modified the signature of ops fgrad, fvar, fuzzy-edge-point?, etc.
%% Modified: Aug. 11,2000
%% Modified: Jan. 26,2001 by Gulsah Cakiroglu
%% Added the theorem
```

```
!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% how to derive fuzzy edge points
%% If grad(I, x, y) = 0  and var(I, x, y) < thrd
%% then (x, y) is an Edge-point.(crisp algorithm)
%% Fuzzy logic:
%% If fgrad(I, x, y) is fuzzy equal to zero and
%% fvar(I, x, y) is fuzzy greater than thrd
%% then (x, y) is a Fuzzy-edge-point.
%% Here zero is fuzzified to fuzzy zero by
%% fzero = tri-fuzzify(zero, dlevel)
%% thrd is also fuzzified to fuzzy thrd by
%% fthrd = tri-fuzzify(thrd, dlevel)
%% Fuzzy and takes a minimum
%% Fuzzy-edge-point is a function sort


spec FUZZY-EDGE is
  import TRI-FUZZIFY

 sort ONE-SORT
 sort Fuzzy-edge-point

  const thrd : Real


  op fgrad : Tri-fuzzy-image, Integer, Integer -> Fuzzy-number
  op fvar : Tri-fuzzy-image, Integer, Integer -> Fuzzy-number
  op fysum : Tri-fuzzy-image, Integer, Integer -> Fuzzy-number
  op felmt : Tri-fuzzy-image, Integer, Integer -> Fuzzy-number
  op fm : Tri-fuzzy-image, Integer, Integer -> Fuzzy-number
  op fsub : Tri-fuzzy-image, Integer, Integer -> Fuzzy-number
  op Fuzzy-edge-point : Tri-fuzzy-image, Integer, Integer -> Uni-intvl


%% Edge-point is a function sort
  sort-axiom Fuzzy-edge-point = Tri-fuzzy-image, Integer, Integer -> Uni-intvl



%% definition of fuzzy Laplacian
  definition of fgrad is
```

```
axiom fgrad is
fa(F:Tri-fuzzy-image, x: Integer, y: Integer)
(fgrad(F, x, y) =
 fuzzy-sub
 (fuzzy-add
  (fuzzy-add(F(plus(x, one), y),
     F(plus(x, inv-plus(one)), y)),
   fuzzy-add(F(x, plus(y, one)),
     F(x, plus(y, inv-plus(one))))),
  fuzzy-add
  (fuzzy-add(F(x, y),F(x, y)),
   fuzzy-add(F(x, y),F(x, y)))))
end-definition

definition of fsub is
axiom fsub is
fa(F:Tri-fuzzy-image, x: Integer, y: Integer)
(fsub(F, x, y) =
 fuzzy-add
 (fuzzy-add
  (fuzzy-add
   (F(x, plus(y, inv-plus(plus(one, one)))),
    F(x, plus(y, inv-plus(one)))),
   fuzzy-add
    (F(x, y),
     F(x, plus(y, one)))),
  F(x, plus(y, plus(one, one)))))
end-definition

definition of fm is
axiom fm is
fa(F:Tri-fuzzy-image, x: Integer, y: Integer)
(fm(F, x, y) =
 (fuzzy-add
  (fuzzy-add
   (fuzzy-add
    (fsub(F, plus(x, inv-plus(plus(one, one))), y),
     fsub(F, plus(x, inv-plus(one)), y)),
    fuzzy-add
    (fsub(F, x, y),
     fsub(F, plus(x, one), y))),
   fsub(F, plus(x, plus(one, one)), y))))
end-definition

definition of felmt is
```

```
  axiom felmt is
  fa(F:Tri-fuzzy-image, x: Integer, y: Integer)
  (felmt(F, x, y) =
   (fuzzy-mult
    (fuzzy-sub(F(x, y),
       F(x, y)),
    fuzzy-sub(F(x, y), F(x, y)))))
  end-definition

  definition of fysum is
  axiom fysum is
  fa(F:Tri-fuzzy-image, x: Integer, y: Integer)
  (fysum(F, x, y) =
   (fuzzy-add
    (fuzzy-add
     (fuzzy-add
      (felmt(F, x, plus(y, inv-plus(plus(one, one)))),
       felmt(F, x, plus(y, inv-plus(one)))),
      fuzzy-add
      (felmt(F, x, y),
       felmt(F, x, plus(y, one)))),
     felmt(F, x, plus(y, plus(one, one))))))
  end-definition

%% definition of fuzzy local variance
  definition of fvar is
  axiom fvar is
  fa(F:Tri-fuzzy-image, x: Integer, y: Integer)
  (fvar(F, x, y) =
   (fuzzy-add
    (fuzzy-add
     (fuzzy-add
      (fysum(F, plus(x, inv-plus(plus(one, one))), y),
       fysum(F, plus(x, inv-plus(one)), y)),
      fuzzy-add
      (fysum(F, x, y),
       fysum(F, plus(x, one), y))),
     fysum(F, plus(x, plus(one, one)), y))))
  end-definition

%% Fuzzy edge point

  definition of Fuzzy-edge-point is
  axiom  Fuzzy-edge-point is
  fa(F:Tri-fuzzy-image, x: Integer, y: Integer, Fep:Fuzzy-edge-point)
  (Fep(F,x,y)=
      (uni-min( fgeq( fvar(F, x, y), tri-fuzzify (thrd, delta)),
                fequal( fgrad(F, x, y), tri-fuzzify (nzero, delta)))))
  end-definition


%%Theorem
  theorem proof is
  fa( x:Integer, y:Integer, I:Image, delta2:Nz ,delta1:Nz)
  (ltx( relax(nnonzero?)(delta2) ,relax(nnonzero?)(delta1)) =>
   ltx(relax(between-zero-one?)
( Fuzzy-edge-point(tri-fuzzify-image(I,delta2),x,y)),
     relax(between-zero-one?)
(Fuzzy-edge-point(tri-fuzzify-image(I,delta1),x,y))))

 end-spec


%% Formal Information Fusion Library Specs
%% Fuzzy set specification
%% Author: Jingsong Li
%% Created: June 20, 1998
%% Last updated: Nov. 3, 1998
%% Comments: Fuzzy set is defined as a total function sort
%% Fuzzy-set = E -> Uni-intvl
%% Modified: Sept. 30, 1999
%% Added components: t-norm, t-conorm
%% Modified: Nov. 28, 1999
%% Added components: t-norm-min, t-conorm-min
%% t-norm-Luka, t-conorm-Luka
%% t-norm-prod, t-conorm-prod
!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% A new sort [0, 1] is defined as Uni-intvl

spec UNI-INTVL is
  import
  translate REAL by {complete-order-lt-2 -> ltx,
     sto-leq -> leqx}
  sort Uni-intvl
  op between-zero-one? : Real -> Boolean
  definition of between-zero-one? is
  axiom between-zero-one? (x) <=>
  leqx(x, one) & leqx(zero, x)
  end-definition

  sort-axiom Uni-intvl = Real | between-zero-one?

end-spec


%% Fuzzy set is defined as a total function.

spec FUZZY-SET is
  import SET, UNI-INTVL

  sort Fuzzy-set

  op alpha-cut : Fuzzy-set, Uni-intvl -> Set
  op height : Fuzzy-set -> Uni-intvl
  op fuzzy-complement : Fuzzy-set -> Fuzzy-set
  op t-norm : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op t-conorm : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op t-norm-min : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op t-conorm-min : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op t-norm-Luka : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op t-conorm-Luka : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op t-norm-prod : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op t-conorm-prod : Fuzzy-set, Fuzzy-set -> Fuzzy-set
  op max : Real, Real -> Real
  op min : Real, Real -> Real

%% Fuzzy-set is a function sort

  sort-axiom Fuzzy-set = E -> Uni-intvl

  definition of fuzzy-complement is
  axiom fa(f1 : Fuzzy-set, f2: Fuzzy-set, d: E)
  (fuzzy-complement(f1) = f2 <=>
   relax(between-zero-one?)(f2(d))
    = plus(one, inv-plus(relax(between-zero-one?)(f1(d)))))
  end-definition


  definition of t-norm is
  axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
```

```
 (t-norm(f1, f2) = f3 <=>
 relax(between-zero-one?) (f3(d)) =
 min(relax(between-zero-one?)(f1(d)),
     relax(between-zero-one?)(f2(d))))
 end-definition

 definition of t-conorm is
 axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
 (t-conorm(f1, f2) = f3 <=>
 relax(between-zero-one?) (f3(d)) =
 max(relax(between-zero-one?)(f1(d)),
     relax(between-zero-one?)(f2(d))))
 end-definition

 axiom fa(x1: Real, x2: Real)
 (min(x1, x2) = x1 <=> leqx(x1, x2) &
  min(x1, x2) = x2 <=> leqx(x2, x1))

 axiom fa(x1: Real, x2: Real)
 (max(x1, x2) = x1 <=> leqx(x2, x1) &
  max(x1, x2) = x2 <=> leqx(x1, x2))

 definition of t-norm-min is
 axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
 (t-norm-min(f1, f2) = f3 <=>
 relax(between-zero-one?) (f3(d)) =
 min(relax(between-zero-one?)(f1(d)),
     relax(between-zero-one?)(f2(d))))
 end-definition

 definition of t-conorm-min is
 axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
 (t-conorm-min(f1, f2) = f3 <=>
 relax(between-zero-one?) (f3(d)) =
 max(relax(between-zero-one?)(f1(d)),
     relax(between-zero-one?)(f2(d))))
 end-definition

 definition of t-norm-Luka is
 axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
 (t-norm-Luka(f1, f2) = f3 <=>
 relax(between-zero-one?) (f3(d)) =
 max(zero, plus(plus(relax(between-zero-one?)(f2(d)),
     relax(between-zero-one?)(f1(d))),
inv-plus(one))))
```

```
  end-definition

  definition of t-conorm-Luka is
  axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
  (t-conorm-Luka(f1, f2) = f3 <=>
  relax(between-zero-one?) (f3(d)) =
  min(one, plus(relax(between-zero-one?)(f2(d)),
relax(between-zero-one?)(f1(d)))))
  end-definition

  definition of t-norm-prod is
  axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
  (t-norm-prod(f1, f2) = f3 <=>
  relax(between-zero-one?) (f3(d)) =
  times(relax(between-zero-one?)(f2(d)),
relax(between-zero-one?)(f1(d))))
  end-definition

  definition of t-conorm-prod is
  axiom fa(f1: Fuzzy-set, f2: Fuzzy-set, f3: Fuzzy-set, d: E)
  (t-conorm-prod(f1, f2) = f3 <=>
  relax(between-zero-one?) (f3(d)) =
  plus(plus(relax(between-zero-one?)(f2(d)),
    relax(between-zero-one?)(f1(d))),
       inv-plus(times(relax(between-zero-one?)(f2(d)),
     relax(between-zero-one?)(f1(d))))))
  end-definition


%% the result of alpha-cut of a fuzzy-set is a crisp set

  definition of alpha-cut is
  axiom fa(f: Fuzzy-set, d: E, S: Set, a: Uni-intvl)
  (alpha-cut(f,a) = S <=> (in(d, S) <=>
   leqx(relax(between-zero-one?)(a),
relax(between-zero-one?)(f(d)))))
  end-definition


  definition of height is
  axiom fa(f: Fuzzy-set, d: E, a: Uni-intvl)
  (height(f) = a <=> leqx(relax(between-zero-one?)(f(d)),
  relax(between-zero-one?)(a)))
  end-definition
```

```
end-spec


%% Formal Information Fusion Library Specs
%% Fuzzy number specification
%% Author: Jingsong Li
%% Created: July 28, 1998
%% Last updated: Nov. 3, 1998
%% Comments: Fuzzy number is a specific fuzzy set when
%% E = Real

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% Fuzzy-number is import from Fuzzy-set by some translation

spec FUZZY-NUMBER is
  import
  translate FUZZY-SET by {Fuzzy-set -> Fuzzy-number,
  Set -> Set-of-Real}

  sort-axiom E = Real

%% Fuzzy-number is normal

  axiom normality is
  fa(f: Fuzzy-number)
  (relax(between-zero-one?)(height(f)) = one)

%% Fuzzy-number is convex


  axiom convexity is
  fa(f: Fuzzy-number, x1: Real, x2: Real, lamd: Uni-intvl)
  (leqx(relax(between-zero-one?)(f(x1)),
relax(between-zero-one?)
(f(plus(times(relax(between-zero-one?)(lamd), x1),
times(plus(one,
   inv-plus
   (relax(between-zero-one?)(lamd))),
      x2)))))
   or
   leqx(relax(between-zero-one?)(f(x2)),
relax(between-zero-one?)
(f(plus(times(relax(between-zero-one?)(lamd), x1),
times(plus(one,
   inv-plus
   (relax(between-zero-one?)(lamd))),
      x2))))))

%% alpha-cut of Fuzzy-number is a closed interval

  theorem alpha-cut-closed is
  fa(f: Fuzzy-number, a: Uni-intvl, S: Set-of-Real)
  (alpha-cut(f, a) = S <=> (fa(d: Real) (in(d, S) =>
  ex(d1: Real, d2: Real)
  leqx(d1, d2) &
  leqx(d1, d) &
  leqx(d, d2))))


 end-spec


%% Formal Information Fusion Library Specs
%% Fuzzy arithmetic operations specification: FUZZY-ARITHM
%% Author: Jingsong Li
%% Created: July 29,1998
%% Last updated: Nov. 10,1998
%% Comments: Definition of these operations is specified
%% following the second method suggested by Klir
%% Modified: Feb. 17,1999
%% Modified: Jan. 12,2001 by Gulsah Cakiroglu
%% Added components: fuzzy-min, fuzzy-max



!! in-package("SPEC")
!! in-grammar('ispec-grammar)

spec FUZZY-ARITHM is

  import  FUZZY-NUMBER

  op fuzzy-add : Fuzzy-number, Fuzzy-number -> Fuzzy-number
  op fuzzy-sub : Fuzzy-number, Fuzzy-number -> Fuzzy-number
  op fuzzy-mult : Fuzzy-number, Fuzzy-number -> Fuzzy-number
  op fuzzy-div : Fuzzy-number, Fuzzy-number -> Fuzzy-number
  op fuzzy-min : Fuzzy-number, Fuzzy-number -> Fuzzy-number
  op fuzzy-max : Fuzzy-number, Fuzzy-number -> Fuzzy-number


%%
%% definition for fuzzy-add
%%
  definition of fuzzy-add is
  axiom
  fuzzy-add (f1: Fuzzy-number, f2: Fuzzy-number)(z:Real)= a:Uni-intvl <=>
  fa(x: Real, y: Real)
  ((z = plus (x,y) =>(leqx(relax(between-zero-one?)(f1(x)),
   relax(between-zero-one?)(a))
   or
   leqx(relax(between-zero-one?)(f2(y)),relax(between-zero-one?)(a)))
   &
  ex(x: Real, y: Real)(z = plus(x,y)
   &
  f1(x) = a & leqx(relax(between-zero-one?)(a),
   relax(between-zero-one?)(f2(y)))
   or
  f2(y) = a & leqx(relax(between-zero-one?)(a),
   relax(between-zero-one?)(f1(x))))))
  end-definition

%%
%% definition for fuzzy-sub
%%

   definition of fuzzy-sub is
  axiom
  fuzzy-sub(f1: Fuzzy-number, f2: Fuzzy-number)(z:Real)= a:Uni-intvl <=>
  fa(x: Real, y: Real)
  ((z =plus (x,inv-plus(y)) =>(leqx(relax(between-zero-one?)(f1(x)),
    relax(between-zero-one?)(a))
   or
   leqx(relax(between-zero-one?)(f2(y)),relax(between-zero-one?)(a))))
   &
  ex(x: Real, y: Real)(z =plus(x,inv-plus(y))
   &
   f1(x) = a & leqx(relax(between-zero-one?)(a),
    relax(between-zero-one?)(f2(y)))
   or
   f2(y) = a & leqx(relax(between-zero-one?)(a),
    relax(between-zero-one?)(f1(x)))))
  end-definition
```

```
%%
%% definition for fuzzy-mult
%%

   definition of fuzzy-mult is
  axiom
  fuzzy-mult (f1: Fuzzy-number, f2: Fuzzy-number)(z:Real)= a:Uni-intvl <=>
  fa(x: Real, y: Real)
  ((z = times (x,y) =>(leqx(relax(between-zero-one?)(f1(x)),
    relax(between-zero-one?)(a))
   or
   leqx(relax(between-zero-one?)(f2(y)),relax(between-zero-one?)(a)))
   &
  ex(x: Real, y: Real)(z = times(x,y)
   &
   f1(x) = a & leqx(relax(between-zero-one?)(a),
  relax(between-zero-one?)(f2(y)))
   or
   f2(y) = a & leqx(relax(between-zero-one?)(a),
    relax(between-zero-one?)(f1(x)))))
  end-definition
%%
%% definition for fuzzy-div
%%
  definition of fuzzy-div is
  axiom
  fuzzy-div (f1: Fuzzy-number, f2: Fuzzy-number)(z:Real)= a:Uni-intvl <=>
  fa(x: Real, y: Real)
  ((x= times (y,z) =>(leqx( relax(between-zero-one?)(f1(x)),
    relax(between-zero-one?)(a))
   or
   leqx( relax(between-zero-one?)(f2(y)),relax(between-zero-one?)(a)))
   &
  ex(x: Real, y: Real)(x = times(y,z)
   &
  f1(x) = a & leqx( relax(between-zero-one?)(a),
    relax(between-zero-one?)(f2(y)))
   or
  f2(y) = a & leqx( relax(between-zero-one?)(a),
    relax(between-zero-one?)(f1(x)))))
  end-definition
```

```
%%
%% definition for fuzzy-min
%%
  definition of fuzzy-min is
  axiom
  fuzzy-min (f1: Fuzzy-number, f2: Fuzzy-number)(z:Real)= a:Uni-intvl <=>
  fa(x: Real, y: Real)
  ((z = min(x,y) =>(leqx(relax(between-zero-one?)(f1(x)),
   relax(between-zero-one?)(a))
   or
   leqx(relax(between-zero-one?)(f2(y)),relax(between-zero-one?)(a)))
   &
  ex(x: Real, y: Real)(z = min(x,y)
   &
  f1(x) = a & leqx(relax(between-zero-one?)(a),
   relax(between-zero-one?)(f2(y)))
   or
  f2(y) = a & leqx(relax(between-zero-one?)(a),
   relax(between-zero-one?)(f1(x))))))
  end-definition

%%
%% definition for fuzzy-max
%%
  definition of fuzzy-max is
  axiom
  fuzzy-max (f1: Fuzzy-number, f2: Fuzzy-number)(z:Real)= a:Uni-intvl <=>
  fa(x: Real, y: Real)
  ((z = max(x,y) =>(leqx(relax(between-zero-one?)(f1(x)),
   relax(between-zero-one?)(a))
   or
   leqx(relax(between-zero-one?)(f2(y)),relax(between-zero-one?)(a)))
   &
  ex(x: Real, y: Real)(z = max(x,y)
   &
  f1(x) = a & leqx(relax(between-zero-one?)(a),
   relax(between-zero-one?)(f2(y)))
   or
  f2(y) = a & leqx(relax(between-zero-one?)(a),
   relax(between-zero-one?)(f1(x))))))
  end-definition
 end-spec


%% Formal Information Fusion Library Specs
%% Fuzzy reasoning operations specification: FUZZY-REASONING
%% Author: Jingsong Li
%% Created: Sept. 29, 1999
%% Last updated: Dec. 10, 1999
%% Comments: Definition of these operations is specified
%% following the method suggested by Kruse

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% Fuzzy reasoning operators coould be defined in various
%% ways. Here I present one version(Lukasiewicz) for
%% fuzzy equal, fuzzy less than and fuzzy great than

spec FUZZY-REASONING is
  import FUZZY-ARITHM


  op fgeq : Fuzzy-number, Fuzzy-number -> Uni-intvl
  op fleq : Fuzzy-number, Fuzzy-number -> Uni-intvl
  op fequal : Fuzzy-number, Fuzzy-number ->  Uni-intvl
  op inf : Real -> Uni-intvl


%% definition of fuzzy equal
 definition of fequal is
  axiom
  fa(f1: Fuzzy-number, f2: Fuzzy-number, x: Real)
  fequal(f1, f2)=
  inf(plus(one,
   inv-plus(abs(plus(relax(between-zero-one?)(f1(x)),
      inv-plus(relax(between-zero-one?)(f2(x)))))))
  end-definition


%% definition of fuzzy greater than and equal
  definition of fgeq is
  axiom
  fa(f1: Fuzzy-number, f2: Fuzzy-number, x: Real)
  fgeq(f1, f2)=
  inf(min(one,
  plus(plus(one,
    relax(between-zero-one?)(f1(x))),
       inv-plus(relax(between-zero-one?)(f2(x))))))
  end-definition
```

```
%% definition of fuzzy less than and equal
  definition of fleq is
  axiom
  fa(f1: Fuzzy-number, f2: Fuzzy-number, x: Real)
  fleq(f1, f2)=
  inf(min(one,
  plus(plus(one,
    relax(between-zero-one?)(f2(x))),
       inv-plus(relax(between-zero-one?)(f1(x))))))
  end-definition


 end-spec



%% Formal Information Fusion Library Specs
%% Image specification: IMAGE
%% Author: Jingsong Li
%% Created: Aug.1, 1998
%% Last updated: Nov. 20, 1999
%% Comments: Image is defined as a function sort
%% Image = Integer, Integer -> Real


!! in-package("SPEC")
!! in-grammar('ispec-grammar)

spec IMAGE is
  import INTEGER, REAL
  sort Image
%% Image is a function sort
  sort-axiom Image = Integer, Integer -> Real

 end-spec



%% Formal Information Fusion Library Specs
%% Fuzzification specification: FUZZIFICATION
%% Author: Jingsong Li
%% Created: Aug.1, 1998
%% Last updated: Nov. 20, 1998
%% Comments: General fuzzification spec is the colimit of
%% IMAGE, FUZZY-ARITHM
%% Last updated: Dec. 10, 1999
%% Comments: General fuzzification spec is the colimit of
%% IMAGE, FUZZY-REASONING

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

spec ONE-SORT is
  sort X
 end-spec

%% General fuzzification
spec FUZZIFICATION is
  import
  translate
  colimit of
  diagram
  nodes ONE-SORT, IMAGE, FUZZY-REASONING
  arcs
  ONE-SORT -> IMAGE : {X -> Real},
  ONE-SORT -> FUZZY-REASONING : {X -> Real}
  end-diagram
  by{fuzzy-reasoning-Nonzero-2 -> Nz,
     fuzzy-reasoning-zero-2 -> nzero,
     fuzzy-reasoning-one-2 -> none,
     fuzzy-reasoning-plus-2 -> nplus,
     fuzzy-reasoning-inv-plus-2 -> ninv-plus,
     fuzzy-reasoning-times-2 -> ntimes,
     fuzzy-reasoning-nonzero?-2 -> nnonzero?}
 end-spec



%% Formal Information Fusion Library Specs
%% Triangular Fuzzification specification: TRI-FUZZIFY
%% Author: Jingsong Li
%% Created: Feb.1, 1999
%% Last updated: July. 20, 1999
%% Comments: Triangular fuzzification method is specified
%% Modified: Oct. 10, 1999
%% Added components: fuzzify : a Larger-than label
%% Modified: Oct. 26, 1999
%% Added components: fuzzify-1 : a Smaller-than label
%% Modified: July 8, 2000
%% Added an op: tri-fuzzify-image to map Image to Tri-fuzzy-image

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% Triangular fuzzification version
%% op tri-fuzzify is to generate a triangular membership
%% function for each crisp value, define an Around label.
%% UnceraintY level is given as a const
%% op tri-fuzzify-2 deals with the situation when
%% uncertainty level is zero
spec TRI-FUZZIFY is
  import FUZZIFICATION

  sorts Tri-fuzzy-image

  op uni-min : Uni-intvl, Uni-intvl -> Uni-intvl
  op fuzzify : Real, Nz -> Fuzzy-number
  op fuzzify-1 : Real, Nz, -> Fuzzy-number
  op tri-fuzzify : Real, Nz -> Fuzzy-number
  op tri-fuzzify-2 : Real -> Fuzzy-number
  op tri-fuzzify-image : Image, Nz -> Tri-fuzzy-image

%% delta is the uncertainty level of the fuzzifying.
  const delta : Nz

  sort-axiom Tri-fuzzy-image = Integer, Integer -> Fuzzy-number

   axiom fa(u1: Uni-intvl, u2: Uni-intvl)
  (uni-min(u1, u2) = u1 <=>
   leqx(relax(between-zero-one?)(u1),
       relax(between-zero-one?)(u2))
   or
   uni-min(u1, u2) = u2 <=>
   leqx(relax(between-zero-one?)(u2),
       relax(between-zero-one?)(u1)))

  definition of tri-fuzzify is
  axiom fa(e: Real, f: Fuzzy-number)
  (tri-fuzzify(e, delta) = f <=>
   (fa(x: Real)
    (ltx(x,
 nplus(e,
       ninv-plus(relax(nnonzero?)(delta)))
     or
     ltx(nplus(e, relax(nnonzero?)(delta)), x)
     <=>
     relax(between-zero-one?)(f(x))= nzero)
    or
    (leqx(nplus(e, ninv-plus(relax(nnonzero?)(delta))), x) &
     leqx(x, e) <=>
     relax(between-zero-one?)(f(x)) =
     div(nplus(x,
       ninv-plus(nplus(e,
       ninv-plus(relax
(nnonzero?)(delta)))),
delta))
    or
    (leqx(e, x) &
     leqx(x, nplus(e, relax(nnonzero?)(delta))) <=>
     relax(between-zero-one?)(f(x)) =
     div(nplus(nplus(e,
     relax(nnonzero?)(delta)),
       ninv-plus(x)),
delta)))

 end-definition

 definition of tri-fuzzify-2 is
 axiom fa(e: Real, f: Fuzzy-number)
 (tri-fuzzify-2(e) = f <=>
  (fa(x: Real)
    (ltx(x, e)
     or
     ltx(e, x)
     <=>
     relax(between-zero-one?)(f(x))= nzero)
    or
    (leqx(x, e)
     &
     leqx(e, x)
     <=>
     relax(between-zero-one?)(f(x))= none)))
 end-definition

%% definition of fuzzification of Image
%% axiom fa(I: Image, x: Integer, y: Integer,
%% F: Tri-fuzzy-image)
%% (F(x, y) = tri-fuzzify(I(x, y), delta))
 definition of tri-fuzzify-image is
```

```
  axiom fa(I: Image, F: Tri-fuzzy-image)
  (tri-fuzzify-image(I, delta) = F <=>
   fa(x: Integer, y: Integer)
   (F(x, y) = tri-fuzzify(I(x, y), delta)))
  end-definition

%% axioms for fuzzify and fuzzify-l

  definition of fuzzify is
  axiom fa(e: Real, f: Fuzzy-number, x: Real)
  (fuzzify(e, delta)(x) = tri-fuzzify(e, delta)(x) <=>
   ltx(x, e)
    or
    relax(between-zero-one?)(fuzzify(e, delta)(x)) = none <=>
    leqx(e, x))
  end-definition


  definition of fuzzify-l is
  axiom fa(e: Real, f: Fuzzy-number, x: Real)
  (fuzzify(e, delta)(x) = tri-fuzzify(e, delta)(x) <=>
   ltx(e, x)
   or
   relax(between-zero-one?)(fuzzify(e, delta)(x)) = none <=>
   leqx(x, e))

  end-definition

 end-spec




%% Formal Information Fusion Library Specs
%% Trapezoidal Fuzzification specification: TRa-FUZZIFY
%% Author: Jingsong Li
%% Created: Feb.16, 1999
%% Last updated: July. 21, 1999
%% Comments: Trapezoidal fuzzification method is specified
%% Modified: July 8, 2000
%% Added an op: tra-fuzzify-image to map Image to Tra-fuzzy-image
```

```
!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% Trapezoidal fuzzification version
%% op tra-fuzzify is to generate a trapezoidal membership
%% function for each crisp value, define an Around label.
%% Uncerainty level is given as two consts
%% op tra-fuzzify-2 deals with the situation when
%% uncertainty level is zero

spec TRA-FUZZIFY is
  import FUZZIFICATION

  sorts Tra-fuzzy-image

  op tra-fuzzify : Real, Nz, Nz -> Fuzzy-number
  op tra-fuzzify-2 : Real -> Fuzzy-number
  op tra-fuzzify-image : Image, Nz, Nz -> Tra-fuzzy-image

%% delta1 + delta2 is the uncertainty level of the fuzzifying.
  const delta1 : Nz
  const delta2 : Nz

  sort-axiom Tra-fuzzy-image = Integer, Integer -> Fuzzy-number

  definition of tra-fuzzify is
  axiom fa(e: Real, f: Fuzzy-number)
  (tra-fuzzify(e, delta1, delta2) = f <=>
   (fa(x: Real)
    (ltx(x,
 nplus(e,
      ninv-plus(nplus(relax(nnonzero?)(delta1),
      relax(nnonzero?)(delta2)))))

     or
     ltx(nplus(e, nplus(relax(nnonzero?)(delta1),
relax(nnonzero?)(delta2))), x)
     <=>
     relax(between-zero-one?)(f(x))= nzero)
    or
    (leqx(nplus(e, ninv-plus(relax(nnonzero?)(delta1))), x) &
     leqx(x, nplus(e, relax(nnonzero?)(delta1))) <=>
     relax(between-zero-one?)(f(x))= none)
    or
    (leqx(nplus(e, ninv-plus(nplus(relax(nnonzero?)(delta1),
    relax(nnonzero?)(delta2)))),
   x) &
      leqx(x, nplus(e, ninv-plus(relax(nnonzero?)(delta1))))
      <=>
      relax(between-zero-one?)(f(x)) =
      div(nplus(x,
        ninv-plus(nplus(e,
        ninv-plus(nplus
        (relax(nnonzero?)(delta1),
relax(nnonzero?)(delta2)))))),
 delta2))
     or
     (leqx(nplus(e, relax(nnonzero?)(delta1)), x) &
      leqx(x, nplus(e, nplus(relax(nnonzero?)(delta1),
     relax(nnonzero?)(delta2))))
      <=>
      relax(between-zero-one?)(f(x)) =
      div(nplus(nplus(e,
      nplus(relax(nnonzero?)(delta1),
    relax(nnonzero?)(delta2))),
        ninv-plus(x)),
 delta2))))

  end-definition

  definition of tra-fuzzify-2 is
  axiom fa(e: Real, f: Fuzzy-number)
  (tra-fuzzify-2(e) = f <=>
   (fa(x: Real)
    (ltx(x, e)
     or
     ltx(e, x)
     <=>
     relax(between-zero-one?)(f(x))= nzero)
    or
    (leqx(x, e)
     &
     leqx(e, x)
     <=>
     relax(between-zero-one?)(f(x))= none)))
  end-definition

%% definition of fuzzification of Image
%% axiom fa(I: Image, x: Integer, y: Integer,
%% F: Tra-fuzzy-image)
%% (F(x, y) = tra-fuzzify(I(x, y), delta1, delta2))

  definition of tra-fuzzify-image is
  axiom fa(I: Image, F: Tra-fuzzy-image)
  (tra-fuzzify-image(I, delta1, delta2) = F <=>
   fa(x: Integer, y: Integer)
   (F(x, y) = tra-fuzzify(I(x, y), delta1, delta2)))
  end-definition

 end-spec



%% Formal Information Fusion Library Specs
%% Guassian Fuzzification specification: GUASS-FUZZIFY
%% Author: Jingsong Li
%% Created: June 10, 1999
%% Last updated: July. 21, 1999
%% Comments: Guassian fuzzification method is specified
%% Modified: July 8, 2000
%% Added an op : guass-fuzzify-image to map Image to Guass-fuzzy-image

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% Guassian fuzzification version
%% op guass-fuzzify is to generate a Guassian membership
%% function for each crisp value, define an Around label.
%% Uncerainty level is given as two consts
%% op guass-fuzzify-2 deals with the situation when
%% uncertainty level is zero

spec GUASS-FUZZIFY is
  import FUZZIFICATION

  sorts Guass-fuzzy-image

  op exp : Real -> Real
  op square : Real -> Real

  op guass-fuzzify : Real, Nz, Nz -> Fuzzy-number
  op guass-fuzzify-2 : Real -> Fuzzy-number
  op guass-fuzzify-image : Image, Nz, Nz -> Guass-fuzzy-image

%% delta is the uncertainty level of the fuzzifying.
```

```
  const sigma : Nz
  const mu : Nz

  sort-axiom Guass-fuzzy-image = Integer, Integer -> Fuzzy-number

  definition of guass-fuzzify is
  axiom fa(e: Real, f: Fuzzy-number)
  (guass-fuzzify(e, sigma, mu) = f <=>
   (fa(x: Real)
    f(x)= exp(ninv-plus(square
(div
 (nplus(e,
ninv-plus(relax(nnonzero?)(mu))),
  relax(nnonzero?)(sigma)))))))
  end-definition

  definition of guass-fuzzify-2 is
  axiom fa(e: Real, f: Fuzzy-number)
  (guass-fuzzify-2(e) = f <=>
   (fa(x: Real)
    (ltx(x, e)
     or
     ltx(e, x)
     <=>
     relax(between-zero-one?)(f(x))= nzero)
    or
    (leqx(x, e)
     &
     leqx(e, x)
     <=>
     relax(between-zero-one?)(f(x))= none)))
  end-definition

%% definition of fuzzification of Image
%% axiom fa(I: Image, x: Integer, y: Integer,
%% F: Guass-fuzzy-image)
%% (F(x, y) = guass-fuzzify(I(x, y), sigma, mu))

  definition of guass-fuzzify-image is
  axiom fa(I: Image, F: Guass-fuzzy-image)
  (guass-fuzzify-image(I, sigma, mu) = F <=>
   fa(x: Integer, y: Integer)
   (F(x, y) = guass-fuzzify(I(x, y), sigma, mu)))
  end-definition
 end-spec


%% Formal Information Fusion Library Specs
%% Defuzzification specification: DEFUZZIFICATION
%% Author: Jingsong Li
%% Created: July 6, 1999
%% Last updated: Sept. 24, 1999
%% Comments: defuzzification method is specified

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% One version of defuzzification
%% defuzzify(F) = (inf(M) + sup(M))/2
%% M=an interval [z] s.t. F(z) = height(F) = one
%% the height of fuzzy number is one


spec DEFUZZIFICATION is
  import FUZZY-NUMBER

  sort Intvl
%% sort Intvl is a subsort of Set-of-Real
  sort-axiom Intvl = Set-of-real

  op defuzzify-1 : Fuzzy-number -> Real
  op inf : Intvl -> Real
  op sup : Intvl -> Real
  op height_intvl: Fuzzy-number -> Intvl
  op div : Real, Real -> Real


  definition of defuzzify-1 is
  axiom fa(F: Fuzzy-number)
  defuzzify-1(F) = div(plus(inf(height_intvl(F)),
    sup(height_intvl(F))),
       plus(one,one))
  end-definition

  Definition of inf is
  axiom fa(I: Intvl, a: Real, x: Real)
  (inf(I)=a
   <=>
   (in(x, I)
    => leqx(a, x)))
  end-definition

  definition of sup is
  axiom fa(I: Intvl, a: Real, x: Real)
  (sup(I)=a <=>
   (in(x, I)
    => leqx(x, a)))
  end-definition

  definition of height_intvl is
  axiom fa(F: Fuzzy-number, I: Intvl, x: Real)
  (height_intvl(F) = I <=>
   (relax(between-zero-one?)(F(x)) = one <=> in(x, I)))
  end-definition

 end-spec



%% Formal Information Fusion Library Specs
%% Defuzzification specification: DEFUZZIFICATION-2
%% Author: Jingsong Li
%% Created: Sept. 24, 1999
%% Last updated: Sept. 24, 1999
%% Comments: defuzzification method is specified

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%% One version of defuzzification
%% defuzzify-2(F) = (inf(M) + sup(M))/2
%% M=an interval [z] s.t. F(z) = alpha-cut of F

spec DEFUZZIFICATION-2 is
  import FUZZY-NUMBER

  sort Intvl
%% sort Intvl is a subsort of Set-of-Real
  sort-axiom Intvl = Set-of-real

  op defuzzify-2 : Fuzzy-number -> Real
  op inf : Intvl -> Real
  op sup : Intvl -> Real
```

```
    op alpha_intvl: Fuzzy-number -> Intvl
    op div : Real, Real -> Real

    const alpha :  Uni-intvl

    definition of defuzzify-2 is
    axiom fa(F: Fuzzy-number)
    (defuzzify-2(F) = div(plus(inf(alpha_intvl(F)),
                                sup(alpha_intvl(F))),
                           plus(one,one)))
    end-definition

    definition of inf is
    axiom fa(I: Intvl, a: Real, x: Real)
    (inf(I)=a
     <=>
     (in(x, I)
      => leqx(a, x)))
    end-definition

    definition of sup is
    axiom fa(I: Intvl, a: Real, x: Real)
    (sup(I)=a <=>
     (in(x, I)
      => leqx(x, a)))
    end-definition

    definition of alpha_intvl is
    axiom fa(F: Fuzzy-number, I: Intvl, x: Real)
    (alpha_intvl(F) = I <=>
     (F(x) = alpha <=> in(x, I)))
    end-definition

end-spec


%% Formal Information Fusion Library Specs
%% Edge point specification : EDGE-POINT
%% Author: Jingsong Li
%% Created: Aug. 2,1998
%% Last updated: Nov. 22,1998
%% Comments: Laplacian-based algorithm is used to derive edge point
%% Edge-point?(x, y) <=> grad(x, y) = 0 ^ var(x, y) > thrd
%% Modified: Jan. 12,2001 by Gulsah Cakiroglu
%% Modified the signiture of ops grad, var, edge-point?, etc.


!! in-package("SPEC")
!! in-grammar('ispec-grammar)

spec EDGE-POINT is
  import IMAGE

  sort Edge-point

  const thrd : Real
  op edge-point? : Image, Integer, Integer -> Boolean
  op grad : Image, Integer, Integer -> Real
  op var : Image, Integer, Integer ->  Real
  op ysum : Image, Integer, Integer ->  Real
  op elmt : Image, Integer, Integer ->  Real
  op m : Image, Integer, Integer ->  Real
  op sub : Image, Integer, Integer ->  Real

  sort-axiom Edge-point = (Image, Integer, Integer) | edge-point?


%% definition of Laplacian
  definition of grad is
  axiom grad is
  fa(I:Image, x: Integer, y: Integer)
  (grad(I, x, y) =
   plus
   (plus
    (plus(I(plus(x, one), y),
  I(plus(x, inv-plus(one)), y)),
     plus(I(x, plus(y, one)),
  I(x, plus(y, inv-plus(one))))),
    inv-plus(plus
     (plus(I(x, y),I(x, y)),
      plus(I(x, y),I(x, y))))))
  end-definition

  definition of sub is
  axiom sub is
  fa(I:Image, x: Integer, y: Integer)
  (sub(I, x, y) =
   plus
   (plus
  (plus
   (I(x, plus(y, inv-plus(plus(one, one)))),
    I(x, plus(y, inv-plus(one)))),
   plus
    (I(x, y),
     I(x, plus(y, one)))),
  I(x, plus(y, plus(one, one)))))
end-definition

definition of m is
axiom m is
fa(I:Image, x: Integer, y: Integer)
(m(I, x, y) =
 (plus
  (plus
   (plus
    (sub(I, plus(x, inv-plus(plus(one, one))), y),
     sub(I, plus(x, inv-plus(one)), y)),
    plus
    (sub(I, x, y),
     sub(I, plus(x, one), y))),
   sub(I, plus(x, plus(one, one)), y))))
end-definition

definition of elmt is
axiom elmt is
fa(I:Image, x: Integer, y: Integer)
(elmt(I,x, y) =
 (times (plus(I(x, y), inv-plus(m(I,x, y))),
         plus(I(x, y), inv-plus(m(I,x, y))))))
 end-definition

definition of ysum is
axiom ysum is
fa(I:Image, x: Integer, y: Integer)
(ysum(I, x, y) =
 (plus
  (plus
   (plus
    (elmt(I, x, plus(y, inv-plus(plus(one, one)))),
     elmt(I, x, plus(y, inv-plus(one)))),
    plus
    (elmt(I, x, y),
     elmt(I, x, plus(y, one)))),
   elmt(I, x, plus(y, plus(one, one))))))
  end-definition

%% definition of local variance
  definition of var is
  axiom var is
  fa(I:Image, x: Integer, y: Integer)
  (var(I, x, y) =
   (plus
    (plus
     (plus
      (ysum(I, plus(x, inv-plus(plus(one, one))), y),
       ysum(I, plus(x, inv-plus(one)), y)),
      plus
      (ysum(I, x, y),
       ysum(I, plus(x, one), y))),
     ysum(I, plus(x, plus(one, one)), y))))
  end-definition

%% definition of edge-point
  definition of edge-point? is
  axiom edge-point?(I, x, y) <=>
  (grad(I, x, y) =  zero &
   complete-order-lt-2(thrd, var(I, x, y)))
  end-definition

 end-spec



%% Formal Information Fusion Library Specs
%% Edge point specification : EDGE
%% Author: Jingsong Li
%% Created: Aug. 2,1998
%% Last updated: Nov. 22,1998

!! in-package("SPEC")
!! in-grammar('ispec-grammar)

%%
%% specification for EDGE
%%
spec EDGE is
  import EDGE-POINT
  sort Edge
  op edge? : Image, Integer, Integer, Integer, Integer -> Boolean
  sort-axiom Edge = (Image, Integer, Integer, Integer, Integer) | edge?

%% definition of edge
  definition of edge? is
    axiom edge?(I, x1, y1, x2, y2) <=>
    (edge-point?(I, x1, y1) &
     edge-point?(I, x2, y2) &
     ((y2 = y1 &
       (fa (x: Integer) (leq(x, x2) &
 lt(x1, x) =>
 edge-point?(I, x, y1))))
      or
      (x2 = x1 &
       (fa (y: Integer) (leq(y, y2) &
 lt(y1, y) =>
 edge-point?(I, x1, y))))
      or
      (fa (x: Integer, y: Integer)
       (lt(y, y2) &
leq(y1, y) &
lt(x, x2) &
leq(x1, x) =>
edge-point?(I, plus(x, one), plus(y, one))))))
    end-definition
 end-spec
```

Chapter 10

Appendix C: Code for Chapter 5

```
function Y=wpd(x)

%-------------------------------------------------------------%
%          WPD: Wavelet Packet Decomposition                  %
%          By Zbigniew Korona                                 %
%          April 1995                                         %
%                                                             %
% Purpose: This program uses a Wavelet Packet Decomposition   %
%          (WPD) algorithm to transform signal "x[]" from time %
%          domain into time/frequency domain Y[][].           %
%-------------------------------------------------------------%

global NLEV LOF HIF  %3 ... ...
len=length(x);
Y=zeros(len,NLEV);
Y(:,1)=x';
for l=1:NLEV-1
   p=l+1;
   nb=2^(l-1)-1;
   for b=0:nb
      q = 2*b;
      r=rem(b,2);
      i = ixlb(l,b);
      a=circonv(Y(i,l),LOF,1);
      Y(ixlb(p,q+r),p)= a(1:2:length(a));
      a=circonv(shift(Y(i,l),1),HIF(length(HIF):-1:1),-1);
      Y(ixlb(p,q+1-r),p)=a(1:2:length(a));
   end
end

% end{wpd}
```

```
function statt
% simulation with one noise level

clear all;

global SIGNUM RELATFI SIGLEN NOISE NCLASS TSIGNUM

tic  %start measure time

SIGNUM = 100; % signals in learning database
initdata;
initwpd;

load testsignalsdb.dat;
load testnoisedb.dat;
load trainingdata.dat;

[mdfeatRel,mdbmask]=getMDB(trainingdata)
ddata=dcomsiginMDB(mdbmask,trainingdata);

bestFI=getbestFI(ddata,mdbmask)

SIGNUM = 600; % signals in learning database
initwpd;

[mdfeatRel,mdbmask]=getMDB(testsignalsdb)
testdata=dcomsiginMDB(mdbmask,testsignalsdb);

%bestRel = [1,3,3,3,3;3,1,1,1,3;1,1,1,1,2;1,1,1,1,3;1,1,1,2,3;1,1,1,
3,1;1,1,2,1,3;1,1,2,2,3;2,1,3,3,1;2,2,3,3,3;2,3,3,1,3;2,3,3,3,1;2,3,3,3,3];
bestRel = [1,1,1,1,1;3,3,3,3,1]
testFIonTDB(testdata,testnoisedb,bestRel,mdfeatRel);
toc    %stop measure time

% end{statt}



function [errorm, errors] = sclcom(data,ttdata,tncelem,tnclass)

%--------------------------------------------------------------%
```

```
%         SCLCOM: Script CLassification COMparison for          %
%                 the problem triangle/rectangle recognition  %
%         By Zbigniew Korona                                    %
%         December 1995                                         %
%                                                               %
% Purpose: This program compares the misclassification rates of%
%     the AFBRS (errorm) vs. the MDWC-based method (errors).   %
%     "data" is the reference database. "ttdata" is            %
%     the test database, "tnclass" is the number of classes    %
%     in the test database, "tncelem" is the number of signals %
%     belonging to one class.                                   %
%---------------------------------------------------------------%

global w1m b1m w2m b2m w3m b3m nntrmi
global w1 b1 w2 b2 w3 b3 nntri
global knumber bkindexm bkindex

[bmdfeaturem,bkindexm,bmdfeature,bkindex,knumber]=sfextcom(data);

% feature selection

feattrainm=feattrt(bmdfeaturem);
[w1m,b1m,w2m,b2m,w3m,b3m,nntrm]=trainmdft(feattrainm);

% backpropagation training using AFBRS

nntrmi=round(nntrm);

feattrain=feattrt(bmdfeature);
[w1,b1,w2,b2,w3,b3,nntr]=trainmdft(feattrain);

% backpropagation training using MDWC

nntri=round(nntr);


tbmdfeaturem=bmdfeat(bkindexm,knumber,ttdata,tnclass,tncelem);

% model theory based feature extraction

tbmdfeature=bmdfeat(bkindex,knumber,ttdata,tnclass,tncelem);

% MDWC-based feature extraction

feattestm=featttt(tbmdfeaturem,tnclass,tncelem);
```

```
classificationm=classif(feattestm,w1m,b1m,w2m,b2m,w3m,b3m);
classificationmi=round(classificationm);
errorm=100*sumsqr(classificationmi-nntrmi)/(2*tnclass*tncelem(1))

feattest=featttt(tbmdfeature,tnclass,tncelem);
classification=classif(feattest,w1,b1,w2,b2,w3,b3);
classificationi=round(classification);
errors=100*sumsqr(classificationi-nntri)/(2*tnclass*tncelem(1))

% end{sclcom}



function i = ixlb(l,b)

%-----------------------------------------------------------------------%
%                                                                       %
%          IXLB: IndeXes for Level-l Box-b cells in WPD                 %
%          By Zbigniew Korona                                           %
%          April 1995                                                   %
%                                                                       %
% Purpose: This program calculates indexes "i[]" for a vector created   %
%          from the WPD matrix.                                         %
%                                                                       %
%-----------------------------------------------------------------------%

global FSCALE

n = 2^(FSCALE-l-1);
i = [b*n+1:(b+1)*n];

% end{ixlb}



function bestFI=getbestFI(data,mdbmask)

% relFI is a matrix 243x5 all possible 5-element relations
% selection is a matrix of all possible combinations
% of 5 elements out of signal_length
% data is a matrix of signals
% disp('1<   2=   3>');
% bestFI is a matrix 2x5 representing relation for
% each class of signals
```

```
global SIGNUM NCLASS SIGLEN RELATFI NOISE
disp('checking FI');


%load relFI.dat;
relFI = [1,3,3,3,3;3,1,1,1,3;1,1,1,1,2;1,1,1,1,3;1,1,1,2,3;1,1,1,3,1;1,1,2,
1,3;1,1,2,2,3;2,1,3,3,1;2,2,3,3,3;2,3,3,1,3;2,3,3,3,1;2,3,3,3,3];

k=1;
isFIOK=1;
relSize=size(relFI,1);
bestFI=zeros(NCLASS,RELATFI);
load bestRel.dat;
bestFI=bestRel;
trgindx=1;
rctindx=1;

bestFI(1,:)=relFI(1,:);
bestFI(2,:)=relFI(1,:);
bestGainclass1=0;
bestGainclass2=0;
snrclassif = zeros(SIGNUM,1);
siginclass=SIGNUM/NCLASS;
start=1;
ending=relSize;

snrtodisplay=zeros(SIGNUM,4);
load trainingnoise.dat;
noise = trainingnoise;

for o=1:SIGNUM
   snr(o)=getSNR(data(o,:),noise(o,:));
   snrtodisplay(o,1)=o;
   snrtodisplay(o,2)=snr(o);
end


for i=start:ending %for all FI relations
   j=i-start+1;
   relFI(i,:)

   for class=1:2

      p1=0; p2=0; n1=0; n2=0;pos=0;neg=0;gain=0;
      snrtodisplay(:,class+2)=zeros(SIGNUM,1);
```

```
% consider only two classes of signals
if (class==1)
   lower = 1;
   upper = 50;
else
   lower = 51;
   upper = 100;
end

for sg=lower:upper              %for all signal in the class
   isFIOK=checkFI(data(sg,:),relFI(i,:));
   if (isFIOK==1)
      snrtodisplay(sg,2+class)=snrtodisplay(sg,2+class)+1;
      if (class==1)
         snrclassif(sg)=snrclassif(sg)+1;
      end
   else
      if(class==2)
         snrclassif(sg)=snrclassif(sg)+1;
      end
   end

   [p1,n1,p2,n2,pos,neg]=getClassDistribution(
    isFIOK,class,p1,n1,p2,n2,pos,neg,sg);
end
distrFI(j,:)=[p1,n1,p2,n2]
[entrop,gain]=entropy(distrFI(j,:),pos)

if (class==1)
   if (gain>=bestGainclass1)
      distrFI(j,:)=[p1,n1,p2,n2];
      bestGainclass1=gain;
      bestFITRG(trgindx,:)=relFI(i,:);
      trgindx=trgindx+1;
      bestDistrib1 = snrtodisplay(:,3);
      relIndx(1)=2+j;
   end
else
   if (gain>=bestGainclass2)
      distrFI(j,:)=[p1,n1,p2,n2];
      bestGainclass2=gain;
      bestFIRCT(rctindx,:)=relFI(i,:);
      rctindx=rctindx+1;
      bestDistrib2 = snrtodisplay(:,4);
      relIndx(2)=2+j;
```

```
            end
         end
      end
   end

   bestFITRG
   bestFIRCT
   bestFI=[1,1,1,1,1;3,3,3,3,1];


function [featRel,bastreen]=getMDB(data)

global RELATFI

[discmeastree,pp]=dmtre(data);
[bastree,bastreen,dmeas]=ldbtre(discmeastree);
knumber=RELATFI; % select as many features as the relation has
[kindex,featRel]=mdcom(bastreen,dmeas,pp,knumber);



function [ener, ennorm]=entfem(wpm)

%-------------------------------------------------------------%
%                                                             %
%          ENTFEM: ENergy for Time/Frequency Energy Map       %
%          By Zbigniew Korona                                 %
%          April 1995                                         %
%                                                             %
% Purpose: This program calculates a time/frequency energy map %
%     "ener[][]" for a signal "wpm[][]" represented in        %
%     time/frequency domain (using WPD). "ennorm" is an energy %
%     of the original signal.                                 %
%                                                             %
%-------------------------------------------------------------%

global NLEV
wpm=wpm';
nn=2^NLEV-1;
ener=zeros(NLEV,length(wpm(1,:)));
ennorm=sum(wpm(1,:).^2);                              % energy
for l=1:NLEV
   nb=2^(l-1)-1;
   for b=0:nb
      ener(l,ixlb(l,b))=wpm(l,ixlb(l,b)).^2;
```

```
                                          %ener - matrix size(ener)=size(wpm)
   end
end
%plotdpd(wpm);
% end{entfem}




function [dmtree,pp]=dmtre(data)

%---------------------------------------------------------------%
%                                                               %
%          DMTRE: Discriminant Measure TREe                     %
%          By Zbigniew Korona                                   %
%          April 1995                                           %
%                                                               %
% Purpose: This program calculates discriminant measure for     %
%   each subbasis (dmtree[]) of the WPD matrix relative to      %
%   the reference database (data) using relative entropy        %
%   as a discriminant measure. "pp[][]" is a time/frequency     %
%   map calculated for each class of signals.                   %
%                                                               %
%---------------------------------------------------------------%

global NCLASS NCELEM
global NLEV FSCALE
nn=2^NLEV-1;
lend=2^FSCALE;
pp=zeros(NCLASS,NLEV*lend);
for ic=1:NCLASS
   sener=zeros(NLEV,lend);
   sennorm=0;
   for k=1:NCELEM(ic)
       if ic==1
          y=wpd(data(k,:));
       else
          y=wpd(data((ic-1)*NCELEM(ic-1)+k,:));
       end
       [ener, ennorm]=entfem(y);
       sener=sener+ener;
       sennorm=sennorm+ennorm;
   end
   for l=1:NLEV
```

```
      nb=2^(l-1)-1;
      for b=0:nb
         index=(l-1)*lend+ixlb(l,b);
         pp(ic,index)=sener(l,ixlb(l,b))/sennorm;
      end
   end
end

dmtree=zeros(1,nn);
for l=1:NLEV
   nb=2^(l-1)-1;
   for b=0:nb
      ib=2^(l-1)+b;
      index=(l-1)*lend+ixlb(l,b);
      dmtreep=zeros(size(1:length(index)));
      for i=1:NCLASS-1
         for j=i+1:NCLASS
            df=pp(i,index).*log2(pp(i,index)./pp(j,index));
            dr=pp(j,index).*log2(pp(j,index)./pp(i,index));
            dmtreep=dmtreep+df+dr;
         end
      end
      dmtree(ib)=sum(dmtreep);
   end
end

%end{dmtre}




function y = circconv(x,f,k)

%--------------------------------------------------------------%
%                                                              %
%          CIRCONV: CIRcular CONVolution                       %
%          By Zbigniew Korona                                  %
%          April 1995                                          %
%                                                              %
% Purpose: This program calculates circular convolution "y[]"  %
%   of signal "x[]" with filter "f[]" using shift length "k".  %
%                                                              %
%--------------------------------------------------------------%
```

```
y = f(1)*x;
for i = 2:length(f)
   x = shift(x,k);
   if f(i)
      y = y + f(i)*x;
   end
end

%end{circonv}



function  bmdf=bmdfeat(kindex,knumber,data,nclass,ncelem)

%---------------------------------------------------------------%
%                                                               %
%          BMDFEAT: Best Most Discriminant FEATures             %
%          By Zbigniew Korona                                   %
%          April 1995                                           %
%                                                               %
% Purpose: This program selects "knumber" of most discriminant %
%     wavelet coefficients (bmdf[][]) from the reference        %
%     database (data) using "kindex" (ordered components of     %
%     most discriminant basis according to the discriminant     %
%     power). "nclass" and "ncelem[]" are number of class       %
%     and the number of signals within one class respectively. %
%                                                               %
%---------------------------------------------------------------%

global NLEV FSCALE
nn=2^NLEV-1;
lend=2^FSCALE;
pp=zeros(nclass,NLEV*lend);
bmdf=zeros(length(data(:,1)),knumber);
for ic=1:nclass
   sener=zeros(NLEV,lend);
   sennorm=0;
   for k=1:ncelem(ic)
       if ic==1
          y=wpd(data(k,:));
          y=y(:);
       else
          y=wpd(data((ic-1)*ncelem(ic-1)+k,:));
          y=y(:);
```

```
         end
         for j=1:knumber
            if ic==1
               bmdf(k,j)=y(kindex(j));
            else
               bmdf((ic-1)*ncelem(ic-1)+k,j)=y(kindex(j));
            end
         end
   end
end

% end{bmdfeat}



function [data,noise]=signalgenB(intsgInSNR)

% rlength is a relation order
% signal has at least that length
global SIGLEN NOISELV SIGNUM NCLASS

disp('Generating rectangles');
pause(1);
repeatcount=1;
snginclss = SIGNUM/NCLASS;    %number of signals in one class
snrminvalue=31;
snrmaxvalue=35;
snrange=68;

intsigOK = zeros(intsgInSNR+1,snrmaxvalue);
data = zeros(snginclss,SIGLEN);
rect=zeros(1,SIGLEN);
k=1;
row=2;

for i=1:snrmaxvalue
   intsigOK(1,i)=i;
end

rand('seed',sum(100*clock));
noise=rand(snginclss ,SIGLEN)*NOISELV;
pause(0.1);

while(k~=snginclss+1) %while less then sig in database
   rect = zeros(1,SIGLEN);
```

```
rand('seed',sum(100*clock));
maxsignval=rand(1,1)+snrange;
pause(0.1);

rand('seed',sum(100*clock));
maxvalue=(NOISELV+rand(1,snginclss)*maxsignval);
minsiglength=5;

rand('seed',sum(100*clock));
begining=1+rand(1,1)*(SIGLEN-10);
pause(0.1);

rand('seed',sum(100*clock));
ending=begining+5+rand(1,1)*(SIGLEN-begining-6);
pause(0.1);

begining=round(begining);
ending=round(ending);
maxvalue=round(maxvalue);

for sig=1:1
   for j=1:SIGLEN

      if ( ending(sig)-begining(sig)<minsiglength )
         if (ending(sig)+minsiglength<SIGLEN)
            ending(sig)=begining(sig)+minsiglength;
         else
            begining(sig)=ending(sig)-minsiglength;
         end;
      end;

      if (j>begining(sig) & j<ending(sig))
         rect(sig,j)=maxvalue(sig);
      else
         rect(sig,j)=0;
      end;
   end;
end;

temp(1,:)=rect(1,:)+noise(k,:);

%================================
snr = getSNR(temp(1,:),noise(k,:));
place = round(snr);
```

```
if (place>=snrminvalue & place<=snrmaxvalue)
   notwritesignal=1;
   while(notwritesignal)
      if ( intsigOK(row,place)<1 )
         intsigOK(row,place)=intsigOK(row,place)+1;
         data(k,:)=temp(1,:);
         notwritesignal=0;
         k=k+1;
      end
      row =row+1;
      if (row>intsgInSNR+1)
         notwritesignal=0;
         repeatcount=repeatcount+1;
         if (mod(repeatcount,40)==0)
            snrange=snrange+1
            snr
            intsigOK'
         end
      end
   end
end
row=2;

%================================
rect=zeros(1,128);

end; % while loop end



function [data,noise]=signalgenA(intsgInSNR)

global SIGLEN NOISELV SIGNUM NCLASS

disp('Generating triangles');
pause(1);
repeatcount=1;
snginclss = SIGNUM/NCLASS;    %number of signals in one class
snrminvalue=31;
snrmaxvalue=35;
snrange=78;


intsigOK = zeros(intsgInSNR+1,snrmaxvalue);
```

```
data = zeros(snginclss,SIGLEN);
triang = zeros(1,SIGLEN);
row=2;
j=1;

rand('seed',sum(100*clock));
noise=rand(snginclss ,SIGLEN)*NOISELV;
pause(0.2);

for i=1:snrmaxvalue
   intsigOK(1,i)=i;
end

while(j~=snginclss+1) %while less then sig in database
   triang = zeros(1,SIGLEN);

rand('seed',sum(100*clock));
mval =rand(1,1)+snrange;
maxsignval=mval;

temp =zeros(1,128);

rand('seed',sum(100*clock));
begining=1+rand(1,1)*(SIGLEN/2);
pause(0.1);

rand('seed',sum(100*clock));
ending=begining+5+rand(1,1)*(SIGLEN-begining-6);
pause(0.1);

rand('seed',sum(100*clock));
maxvalue=mval+rand(1,snginclss)*maxsignval;
pause(0.1);

rand('seed',sum(100*clock));
maxloc=begining+rand(1,1)*(SIGLEN-begining-5);
pause(0.1);

begining=round(begining);
ending=round(ending);
maxvalue=round(maxvalue);
value=0;
rand('seed',sum(100*clock));

for i=1:1
```

```
   maxloc(i)=2+begining(i)+rand(1,1)*(ending(i)-begining(i)-4);
   maxloc(i)=round(maxloc(i));
   triang(i,maxloc(i))=maxvalue(i);
end;

for sig=1:1
   lstep=maxvalue(sig)/(maxloc(sig)-begining(sig));
   value=maxvalue(sig);
   for i=maxloc(sig):-1:begining(sig)
      value=value-lstep;
      if (value>0)
         triang(sig,i)=value;
      end;
   end;
   value=maxvalue(sig);
   rstep=maxvalue(sig)/(ending(sig)-maxloc(sig));
   for i=maxloc(sig):ending(sig)
      value=value-rstep;
      if (value>0)
         triang(sig,i)=value;
      end;
   end;
end;

temp(1,:)=triang(1,:)+noise(j,:);

%=================================
snr = getSNR(temp(1,:),noise(j,:));
place = round(snr);

if (place>=snrminvalue & place<=snrmaxvalue)
   notwritesignal=1;
   while(notwritesignal)
      if ( intsigOK(row,place)<1 )
         intsigOK(row,place)=intsigOK(row,place)+1;
         data(j,:)=temp(1,:);
         triang(1,:)=zeros(1,128);
         temp(1,:)=zeros(1,128);
         j=j+1;
         notwritesignal=0;
      end
      row =row+1;
      if (row>intsgInSNR+1)
         notwritesignal=0;
         repeatcount=repeatcount+1;
         if (mod(repeatcount,20)==0)
            snrange = snrange+1
            intsigOK'
         end
      end
   end
end
row=2;

%==================================
triang=zeros(1,128);
end; %while loop end



function plotsignals(signum,data)

sigdispl=25;

for i=1:signum/sigdispl
   figure;
   clf;
   for j=(i-1)*sigdispl+1:i*sigdispl
      subplot(5,5,mod(mod(j-1,sigdispl)+1,sigdispl+1) );
      plot(data(j,:));
   end
   xlabel('Database after decoposition in MDB');
end



function [entrop,gain]=entropy(pn,pos)

% assumption is that for number of classes
% pn has corresponding number of rows
% for k classes it appears     | p1 n1 p2 n2 |  for relation

global NCLASS SIGNUM

p=pos;
n=(SIGNUM/NCLASS)-p;
entrop=1;
```

```
Inp=getInp(p,n)

for i=1:2
   classEntrop(i)=getInp( pn(2*i-1),pn(2*i) );
end

E=((pn(1)+pn(2))*classEntrop(1)+(pn(3)+pn(4))*classEntrop(2))/(n+p);

entrop=E;
gain=abs(Inp-E);
%save enthrop.dat enthrop -ascii;
%save gain.dat gain -ascii;



function generateSignalsDB(signum)

global SIGNUM RELATFI SIGLEN NOISE NCLASS TSIGNUM

initdata;
SIGNUM = signum; % signals in learning database
numofsnr = 10

load testsignalsdb.dat;
load testnoisedb.dat;
data = testsignalsdb;
noise = testnoisedb;

for i=1:600
   snr(i)=getSNR(data(i,:),noise(i,:));
end

snr = sort(snr);

figure
clf
title('SNR Distribution');
plot(snr);
xlabel('Signal');
ylabel('Signal to Noise Ratio');


snr = sort(round(snr));
figure
clf
```

```
title('SNR Distribution II');
plot(snr);
xlabel('Signal');
ylabel('Signal to Noise Ratio');



function data=datagen(intsginSNR)

global NCLASS SIGLEN RELATFI NOISELV SIGNUM

disp('Generate signals database');
pause(1);

snginclss = SIGNUM/NCLASS;    %number of signals in one class

data = zeros(intsginSNR*SIGNUM,SIGLEN);

%[classA,noiseA]=signalgenA(intsginSNR);
[classB,noiseB]=signalgenB(intsginSNR);
save gapsigrect.dat classB -ascii;
save gapnoiserect.dat noiseB -ascii;

data(1:snginclss ,:)=classA;
data((snginclss +1):SIGNUM,:)=classB;
noise(1:snginclss,:)=noiseA;
noise((snginclss+1):SIGNUM,:)=noiseB;

save gapsig.dat data -ascii;
save gapnoise.dat noise -ascii;



function [snrdata,classesDistr]=testFIonTDB(testdata,bestFI,noise)

% relFI is a 5-binary relations
% testdata is a matrix of signals
% disp('1<   2=   3>');

global SIGNUM NCLASS SIGLEN RELATFI

SNR=0;
isFIOK=0;
classApos=0;classBpos=0;
pos=0;      % signals positively recognized by the relation
```

```
classesDistr=zeros(1,NCLASS);
siginclass=SIGNUM/NCLASS;
snrdata = zeros(3,27);


for i=1:NCLASS

   for sg=((i-1)*siginclass+1):(i*siginclass)
      isFIOK=checkFI(testdata(sg,:),bestFI(i,:));

      SNR=getSNR(testdata(sg,:),noise(sg,:));
      column = round(SNR+.51);
      t=[SNR,column];
      snrdata(1,column)=SNR;

      if (isFIOK)
         pos=pos+1;

         if (i==1)
            classApos=classApos+1;
         elseif (i==2)
            classBpos=classBpos+1;
         end
         snrdata(1,sg)=sg;
         snrdata(2,column)=classApos;
         snrdata(3,column)=classBpos;
      end
   end

   classesDistr(i)=pos;
   pos=0;
end

save snrdata.dat snrdata -ascii;
snrdata'
classesDistr


function y = shift(x,k)

%-------------------------------------------------------------%
%          SHIFT: linear SHIFT                                %
%          By Zbigniew Korona                                 %
%          April 1995                                         %
%                                                             %
% Purpose: This program calculates a shifted version "y[]" of  %
%          the input signal "x[]" with a shift length "k".     %
%--------------------------------------------------------------%

[r,c] = size(x);
n = max(r,c);
k = k-n*floor(k/n);
if c==1
   y = [x(k+1:n);x(1:k)];  % up/down for col vector
else
   y = [x(k+1:n),x(1:k)];  % left/right for row vector
end

% end{shift}



function plotdpd(dpd)

%-----------------------------------------------------------------------%
%                                                                       %
%          PLOTDPD: PLOT Display of wavelet Packet Decomposition        %
%          By Zbigniew Korona                                           %
%          April 1995                                                   %
%                                                                       %
% Purpose: This program plots function decomposed by means of WPD       %
%          (dpd) within an adequate decomposition structure.            %
%                                                                       %
%-----------------------------------------------------------------------%

global SIGLEN FSCALE CSCALE LEVELS

nin = nargin;
coarsest = max(CSCALE,FSCALE-7);
dpd = dpd';
save dpd dpd -ascii;

figure;
clf;

x = linspace(0,1,SIGLEN);
bls = '-';     % box line style is solid
blc = 'c';     % box line color is cyan
bhh = 0.5;     % box half height
blw=2;         % box line width
```

```
sbot = coarsest - bhh;
stop = FSCALE + bhh;
cla;
set(gca,'NextPlot','add','Box','on','XLim',axlim(x),'YLim',axlim(
[coarsest-0.5,FSCALE+0.5]));

for l = FSCALE-coarsest+1:-1:1
   s = FSCALE-l+1;
   sb = s-bhh;
   st = s+bhh;
   line([0,1],[sb,sb],'LineStyle',bls,'Color',blc,'LineWidth',blw);
   nb = 2^(l-1);
   for b=1:2:nb-1
      pbox = b/nb;
      line([pbox,pbox],[sbot,st],'LineStyle',bls,'Color',blc,'LineWidth',blw);
   end
end

line([0,1],[st,st],'LineStyle',bls,'Color',blc,'LineWidth',blw);
line([0,0],[sbot,stop],'LineStyle',bls,'Color',blc,'LineWidth',blw);
line([1,1],[sbot,stop],'LineStyle',bls,'Color',blc,'LineWidth',blw);

for l = FSCALE-coarsest+1:-1:1
   mult = 1/(2*max(abs(dpd(l,:))+eps));
   plotwpd(mult*dpd(l,:),x,FSCALE-l+1);
end

ylabel('Resolution Level');
xlabel('Normalized Frequency');

% end{plotdpd}



function [btre,btren,dmeas]=ldbtre(dmtree)

%-----------------------------------------------------------------------%
%                                                                       %
%          LDBTRE: Local Discriminant Basis TREe                       %
%          By Zbigniew Korona                                           %
%          April 1995                                                   %
%                                                                       %
% Purpose: This program finds most discriminant basis based on         %
%          discriminant measure tree (dmtree). "btre[]" is             %
%          a basis tree with most disriminant subbases and             %
%          their successors having value "1", "btren" is a basis       %
%          tree with only most disriminant subbases having value "1",  %
%          "dmeas" is  ordered discriminant measure tree where         %
%          parent nodes have at least as much discriminant power       %
%          as sum of the discriminant power of their two children.     %
%                                                                      %
%----------------------------------------------------------------------%

nn=length(dmtree);
nl=log2(nn+1); %=4
btre=zeros(size(dmtree));
btren=zeros(size(dmtree));
dmeas=dmtree;
nlnod=2^(nl-1);%=8
btre(nlnod:2*nlnod-1)=ones(1,nlnod);
btren(nlnod:2*nlnod-1)=ones(1,nlnod);
for l=nl-1:-1:1
   il=2^(l-1);
   for b=0:il-1              % il-1=3
      ipar=il+b;             % ipar= from 1 to 7
      cpar=dmeas(ipar);      % reals at location ipar
      ilch=2^l+2*b;          % ilch = from 2 to 14
      clch=dmeas(ilch);      % real at location ilch
      irch=2^l+2*b+1;        % irch = from 3 to 15
      crch=dmeas(irch);      % real at location irch
      cbch=clch+crch;
      if cpar>=cbch
         btre(ipar)=1;
         btren(ipar)=1;
         btren(ilch)=0;
         btren(irch)=0;
      else
         dmeas(ipar)=cbch;
      end
   end
end
% end{ldbtre}


function initwpd

global CSCALE HIF LOF FSCALE NLEV NCLASS NCELEM SIGNUM

FSCALE=7;
```

```
CSCALE=4;
NLEV=FSCALE-CSCALE+1;
NCELEM=zeros(size(1:NCLASS));

for jc=1:NCLASS
   NCELEM(jc)=SIGNUM/NCLASS;
end


% the following are parameters used to build a lowpass filter
% using Daubechies_4 wavelet basis

p(1)=(1+sqrt(3));
p(2)=(3+sqrt(3));
p(3)=(3-sqrt(3));
p(4)=(1-sqrt(3));

p=p/norm(p);

% the following is a vector used to build a lowpass filter
% using Daubechies_4 wavelet basis

q=[p(4) -p(3) p(2) -p(1)];

LOF=p;
HIF=q;
% end{initwpd}



function initdata

global NCLASS SIGLEN RELATFI NOISELV SIGNUM TSIGNUM

RELATFI=5;    % 5-nary relatio
NCLASS = 2;   % class of signals
SIGLEN=2^7;   % signals length
NOISELV = 1;  % noise level (max value)



function [p1,n1,p2,n2,pos,neg]=getClassDistribution
(isFIOK,class,p1,n1,p2,n2,pos,neg,sg)

global NCLASS SIGNUM
```

```
if (isFIOK==1)
   pos=pos+1;
else
   neg=neg+1;
end


if (class==1)
   if (isFIOK==1)
      p1=p1+1;
   else
      n2=n2+1;
   end
else    % second class
   if (isFIOK==0)
      p2=p2+1;
   else
      n1=n1+1;
   end
end





function SNRatio=getSNR(sg,noise)

global SIGLEN

signalenergy=sum(sg.^2)/SIGLEN;
noisenergy=sum(noise.^2)/SIGLEN;

temp=10*log10(signalenergy/noisenergy);
if (temp<1)
   SNRatio=0;
else
   SNRatio=temp;
end
```

```
function inp=getInp(p,n)

prob =p/(p+n);

if (p*n>0)
   temp=-( prob*log2(prob) + (1-prob)*log2(1-prob) );
else
   temp=0;
end

if (temp<0)
   inp=0;
else
   inp=temp;
end;
```

```
function ddata=dcomsiginMDB(mdbmask,data)

global NCELEM NCLASS SIGLEN

rows=size(data,1);
cols=size(data,2);
ddata=zeros(rows,cols);
ddrows=1;
for i=1:4
   for j=1:8
      tempDecomp(i,j)=-1;
   end
end

tempDecomp(4,:)=mdbmask(1,8:15);
tempDecomp(3,1:4)=mdbmask(1,4:7);
tempDecomp(2,1:2)=mdbmask(1,2:3);
tempDecomp(1,1)=mdbmask(1,1);
```

```
tempDecomp
mdbmask
lastpos=1;
j=3;
s=1;

for ic=1:NCLASS

   for k=1:NCELEM(ic)
      if ic==1
         y=wpd(data(k,:));
      else
         y=wpd(data((ic-1)*NCELEM(ic-1)+k,:));
      end
      y=y';
      %plotdpd(y);
      decompsig=zeros(1,SIGLEN);
      sidx=1;
      for i=4:-1:1
         coef2pow=2^(i-1);
         for j=1:coef2pow
            if (tempDecomp(i,j)==1)
               subbase=j;
               elnum=SIGLEN/(2^(i-1));
               startel=(subbase-1)*elnum+1;
               endelem=startel+elnum-1;
               decompsig(1,sidx:sidx+elnum-1)=y(i,startel:endelem);
               sidx=sidx+elnum;
            end
         end
      end
      ddatabis(ddrows,:)=decompsig;
      %ddata(ddrows,:)=data(ddrows,:).*decompsig;
      ddata(ddrows,:)=decompsig;
      ddrows=ddrows+1;
   end
end
%save ddata.dat ddata -ascii;



function isFIOK=checkFI(signal,relFI)

% signal is a single signal from the DB
% relFI is a vector 1x5 (single relation to be checked)
% mdbmask is a mask of most discriminant basis

global NCELEM NCLASS SIGLEN RELATFI NOISELV

p=1;                    %place in the signal
pcopy=p;         % when restart form begining of the signal
q=2;                   %fixed place when binary relation satisfied
morechoises=1;
isFIOK=0;
birel=0;               %binary relation
rp=1;                  % relation place 1st, 2nd, ...5th
k=1;
j=1;
for s=1:RELATFI
   mout(s)=-1;
end

count=1;
%checking_rel = relFI

% while there is more elements in subbases to search for relation and
% the relation is not satisfied yet

while ( (isFIOK==0) & (morechoises==1) )

   if (p+1<SIGLEN-rp)

   % birel is binary relation between 2 componerts
   if ( (signal(p)>2) & (abs(signal(p)-signal(q))<0.1*NOISELV))
      birel=2;
   elseif ( (signal(p)+NOISELV)<signal(q) )
      birel=1;
   elseif ( (signal(p)-NOISELV)>signal(q) )
      birel=3;
   end
else
   morechoises=0;
end

   if (relFI(rp)==birel)
      mout(1,rp)=p;
      mout(2,rp)=q;
      rp=rp+1;
      pcopy=p;
      p=q; q=p+1;
      if (rp>RELATFI)
         isFIOK=1;
      end

   else
      q=q+1;
      if (q>128)
         pcopy=p;        % start from successing element of last start
         p=pcopy+1;      % pcopy ensures that we restart searching
         q=p+1;
         rp=1;
      end

      if (p+1>128-rp)
         morechoises=0;
      end
   end
end   % while loop

%save mout.dat mout -ascii;
%save tempout.dat tempout -ascii;



function relFI=buildrelFI(elements)

% passing argument is a vector 1xN
% and N is number of elements in relation
% disp('1<   2=   3>');
pointer = elements;
maxrel=3;
counter=1;
m=1;

for i=1:elements
   location(i)=maxrel;
end;

for i=1:elements
   birelation(i)=1;
   pow(i)=elements-i;
end;

while ( sum(birelation)<elements*maxrel )
```

```
   relFI(m,:)=birelation;
   m=m+1;

   for idx=elements:-1:1
      if ( mod(counter,maxrel^pow(idx))==0 )
         birelation(idx)=1+mod(birelation(idx),maxrel);
      end
   end
   counter=counter+1;
end;

relFI(m,:)=birelation;
function [lims,largest,smallest,range] = axlim(x)

%---------------------------------------------------------------%
%                                                               %
%            AXLIM: AXis LIMits                                 %
%            by Zbigniew Korona                                 %
%            April 1995                                         %
%                                                               %
% Purpose: This program calculates the "value" axis limits      %
%   for function x: lims[2] - lower and upper limit, largest -  %
%   maximum value, smallest - minimum value, range - the        %
%   the difference between the maximum and the minimum.         %
%                                                               %
%---------------------------------------------------------------%

c = 0.025;                         % margin scaling coefficient
largest = max(x);
smallest = min(x);
range = largest - smallest;
if range
margin = c*range;                  % margin
else
margin = c*smallest;
end
lims(1) = smallest - margin;       % lower limit
lims(2) = largest + margin;        % upper limit

%end{axlim}
```

Bibliography

[1] Specware: Language manual, version 2.0.3. Technical report, Kestrel Institute, 1998.

[2] Specware: User guide, version 2.0.3. Technical report, Kestrel Institute, 1998.

[3] Omg unified modeling language specification, version 1.3. Technical report, Object Management Group, 1999.

[4] G. A. Baxes. *Digital Image Processing.* John Wiley & Sons, Inc., 1994.

[5] M. D. Bedworth and J. C. O'Brien. Pittfalls in data fusion (and how to avoid them). In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 437–444, 1999.

[6] C. C. Chang and H. J. Keisler. *Model Theory.* North Holland, Amsterdam, New York, Oxford, Tokyo, 1992.

[7] R. Coifman, R. and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38, no.2:713–718, 1992.

[8] B. V. Dasarathy. *Decision Fusion.* IEEE Computer Society Press, 1994.

[9] B. V. Dasarathy. Sensor fusion potential exploitation - innovative architectures and illustrative applications. *Proceedings of IEEE*, 85, No.1:24–38, 1997.

[10] S. A. DeLoach and M. M. Kokar. Category theory approach to fusion of wavelet-based features. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 117–124, 1999.

[11] F. Galvin. Horn sentences. *Annals of Mathematical Logic*, 1:389–422, 1970.

[12] H. Gao, M. M. Kokar, and J. Weyman. An approach to automation of fusion using specware. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 109–116, 1999.

[13] D. L. Hall. *Mathematical Techniques in Multisensor Data Fusion.* Artech House, Boston - London, 1992.

[14] L. Hong. Multiresolutional filtering using wavelet transform. *IEEE Transactions on Aerospace and Electronic Systems*, 29(4):1244–1251, 1993.

[15] A. Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16:14–21, 1951.

[16] J. R. Jang and C. Sun. Neuro-fuzzy modeling and control. *IEEE Transactions*, 1995.

[17] H. J. Keisler. Reduced products and horn classes. *Trans. Amer. Math. Soc. Ser.2*, 81:307–328, 1965.

[18] G. J. Klir. On the alleged superiority of probabilistic representation of uncertainty. *IEEE Transactions on Fuzzy Systems*, 2, No.1, 1994.

[19] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prentice Hall PTR, 1995.

[20] G. J. Klir and B. Yuan. *Fuzzy Set Theory Foundations and Applications.* Prentice Hall PTR, 1997.

[21] M. M. Kokar, M. D. Bedworth, and K. B. Frankel. A reference model for data fusion systems. In *Sensor Fusion: Architectures, Algorithms, and Applications IV*, pages 191–202. SPIE, 2000.

[22] M. M. Kokar and Z. Korona. A formal approach to the design of feature-based multi-sensor recognition systems. *International Journal of Information Fusion (in print*, 2001.

[23] M. M. Kokar, J. A. Tomasik, and J. Weyman. A formal approach to information fusion. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 133–140, 1999.

[24] Z. Korona and M. M. Kokar. Lung sound recognition using model-theory based feature selection. *Applied Signal Processing*, 5:152–169, 1998.

[25] J. Li, M. M. Kokar, and J. Weyman. Incorporating uncertainty into the formal development of the fusion operator. In *Proceedings of the Second International Conference on Information Fusion, Vol. 1*, pages 125–132, 1999.

[26] J. S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice Hall, Inc., 1990.

[27] R. C. Luo and M. G. Kay. Multisensor integration and fusion in intelligent systems. *IEEE Transactions on Systems, Man and Cybernetics*, 19-5:901–931, 1989.

[28] J. A. Makovsky. Why horn formulas matter in computer science: Initial structure and generic examples. In *CAAP'85 Arbres en Algebre et Programmation 10*, pages 374–385, 1985.

[29] J.M. Manyika and H.F. Durrant-White. An information-theoretic approach to management in decentralized data fusion. In *Sensor Fusion V*, volume 1828, pages 202–213. SPIE, 1992.

[30] M. Markin, C. Harris, M. Bernhardt, J. Austin, M. Bedworth, P. Greenway, R. Johnston, A. Little, and D. Lowe. Technology foresight on data fusion and data processing. *Publication of the Royal Aeronautical Society*, 1997.

[31] E. A. Palyutin, J. Saffe, and S. S. Starchenko. Models of superstable horn theories. *Algebra i Logika*, 24:278–326, 1985.

[32] B. C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, 1991.

[33] R. J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[34] N. Saito. *Local Feature Extraction and Its Applications Using a Library of Bases.* PhD thesis, Yale University, 1994.

[35] A. Shulsky. *Silent Warfare: Understanding the World of Intelligence.* Brassey's, 1993.

[36] Y. V. Srinivas. Category theory: Definitions and examples. Technical Report TR-90-14, University of California at Irvine, 1990.

[37] A. N. Steinberg, C. L. Bowman, and F. E. White. Revisions to the jdl data fusion model. In *Proceedings of the SPIE. Sensor Fusion: Architectures, Algorithms and Applications*, pages 430–441. SPIE, 1999.

[38] S. C. A. Thomopoulos. Dignet: a self-organizing neural network for automatic pattern recognition, classification, and data fusion. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 478–495. SPIE, 1992.

[39] J. Tomasik. On products of neat structures. *Annals of Mathematical Logic*, 36:12–16, 1976.

[40] P. K. Varshney. *Distributed Detection and Data Fusion.* Springer-Verlag, 1996.

[41] J. M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 9:8–24, 1990.

[42] L. A. Zadeh. Fuzzy logic. *Computer*, 21, No. 4:83–93, 1988.